L'AMSTRAD AVEC PLAISIR

Comment faire de bons programmes

R.A. et J.W. PENFOLD



L'AMSTRAD AVEC PLAISIR : COMMENT FAIRE DE BONS PROGRAMMES

Traduction autorisée de l'ouvrage publié en langue anglaise.
Titre original : « An Introduction to Programming the Amstrad CPC 464 »

First published in English october 1984.

Traduction française de Pierre MAURICE

ISBN 2-90-4457-31-3

EDIMICRO
DÉPARTEMENT ÉDITIONS DE F.D.S. SARL
121/127, avenue d'Italie, 75013 Paris

L'AMSTRAD AVEC PLAISIR

COMMENT FAIRE DE BONS PROGRAMMES

par
R.A. et J.W. PENFOLD

Collection « Ordinateurs familiaux »



PRÉFACE

L'Amstrad CPC 464 n'est pas un ordinateur domestique commun. Sa particularité la plus significative est l'adjonction d'un lecteur de cassette et d'un moniteur monochromatique en couleur. L'ensemble constitue un matériel de premier choix, et les avantages d'un lecteur de cassette incorporé sont indéniables ; de même, l'utilisation d'un moniteur au lieu d'un écran de télévision est très profitable. Le fait que l'ensemble soit relativement bon marché ne signifie pas que les performances de l'ordinateur présentent des lacunes. Grâce à ses possibilités graphiques, ses 80 colonnes de texte, et son générateur de son de grande qualité, il est actuellement l'un des meilleurs micro-ordinateurs disponibles.

L'excellente structure matérielle s'accommode parfaitement du langage interne de cet ordinateur : Locomotive Basic, qui offre une vaste gamme de commandes parmi lesquelles on trouve des instructions capables d'exploiter parfaitement le graphisme et les capacités sonores de la machine. Bien qu'un Basic évolué de ce type puisse intimider le débutant, il rend la programmation beaucoup plus aisée lorsque les diverses instructions sont comprises. Ce n'est pas aussi difficile que l'on pourrait le croire à condition d'envisager l'étude progressivement.

SOMMAIRE

Chapitre 1

VARIABLES ET TABLEAUX

Examinez attentivement un jeu informatique classique ; s'il est bien conçu, il comprendra sûrement un score sur l'écran. Au fur et à mesure que vous touchez des extra-terrestres ou tout autre chose, le score progresse. L'ordinateur doit être capable d'enregistrer le score actuel, au besoin de le modifier, puis de mémoriser le nouveau score.

Dans le cadre du langage Basic, le contenu de l'information qui évolue périodiquement tient à des variables. Elles sont distinctes des constantes, nombres valant par exemple 1 100 ou 11 273 qui peuvent figurer à l'intérieur des lignes d'un programme, et qui évidemment ne peuvent changer de valeur.

On donne des noms aux variables que l'on utilise dans un programme. Ces noms ne doivent pas être des mots-clés du Basic, que l'ordinateur reconnaît comme instructions spécifiques. Dans Locomotive Basic, le nom des variables peut comporter des lettres minuscules ou majuscules ou des chiffres, le premier caractère devant être une lettre. Toutes les lettres et tous les nombres utilisés servent à distinguer une variable d'une autre. Beaucoup de Basic n'utilisent que les deux premiers caractères, cela peut causer des difficultés. Cependant, Locomotive Basic ne différencie pas minuscules et majuscules dans un nom de variable. Si le nom de la variable se termine avec le signe « dollar » ($) cela indique qu'il s'agit d'une chaîne de caractères. Ces dernières sont traitées différemment des variables numériques étudiées dans ce chapitre et font l'objet d'un chapitre particulier.

Même si de nombreux programmes utilisent comme noms de variables, des expressions algébriques d'une seule lettre, il est souvent préférable de se servir de noms plus longs qui donnent une meilleure indication de l'utilité de la variable. Par exemple, si une variable indique le score, appelez-la « score ». Cependant l'emploi de X et Y comme noms de

variables, pour repérer un point sur l'écran, est de fait universellement utilisé. Des noms de variables longs prennent cependant plus de mémoire, mais avec l'Amstrad il y a peu de risque de dépassement.

Le Basic accepte des mots tapés en lettres majuscules ou minuscules. Cependant ces mots-clés apparaissent en majuscule lorsqu'un listing du programme est inscrit sur l'écran. Le nom des variables reste toujours tel qu'il a été tapé. C'est pourquoi il est judicieux d'écrire les noms de variables en minuscules de façon à les différencier des mots-clés.

Toutes les opérations mathématiques ou logiques peuvent être réalisées à l'aide de variables, et le résultat des opérations mathématiques peut changer le contenu d'une variable ou bien en créer une nouvelle. Ainsi *score = score + 10* change le contenu de la variable score, *prix = coût + taxe* crée une nouvelle variable prix, contenant la valeur du contenu de *coût* plus le contenu de *taxe*. Si la variable *prix* existait déjà, alors sa valeur aurait été mise à jour.

Il est important de noter que lorsqu'une des variables est utilisée du côté droit d'une affectation, comme dans les exemples ci-dessus, sa valeur n'est pas changée, à moins qu'elle ne figure aussi du côté gauche, et elle n'est pas détruite. Dans l'exemple précédent, *coût* et *taxe* sont inchangés et pourront être utilisés pour d'autres calculs.

A partir de ces exemples, il est clair que les variables utilisées en programmation sont bien différentes des variables algébriques. Une instruction X = X + 10 est un non-sens en algèbre, mais est fréquemment employée en informatique. Il faudra toujours se souvenir qu'une variable est un emplacement repérable dont le contenu peut être modifié. Ainsi la ligne *score = score + 10* signifie en fait : regardons le contenu de l'emplacement mémoire repéré par *score,* ajoutons-lui *10* et rangeons le résultat dans la place mémoire appelée *score.*

Locomotive Basic permet aussi l'emploi de variables entières qui peuvent être utilisées pour ne représenter que des nombres entiers. On les distingue par le dernier caractère de leur nom, qui est le signe pourcentage (%). Les nombres représentés sont compris entre − 32 768 et +

32 767, mais ces variables prennent peu de place mémoire et permettent des opérations plus rapides.

Si une valeur non entière est attribuée à une variable entière, elle est convertie automatiquement en nombre entier. Cela s'effectue par substitution au profit de l'entier immédiatement inférieur. Ainsi 3,3 et 3,897 seraient remplacés par 3, et – 3,53 par – 4 (ce dernier exemple est cause d'insomnies chez bon nombre de programmeurs).

BOUCLES

L'une des principales utilisations de l'ordinateur est la réalisation de tâches répétitives.

Lorsqu'un événement doit se reproduire en quantité fixe, on utilise logiquement une variable pour compter le nombre de fois, où cet événement a eu lieu. En fait, même si cela est particulièrement simple, Basic possède des instructions spéciales réalisant cette opération : FOR, STEP et NEXT et WHILE... WEND.

Les boucles FOR... NEXT sont les plus simples. Elles sont utilisées lorsqu'une action doit être répétée un nombre connu de fois ou dans une plage définie dans l'instruction. On attribue à la variable de contrôle, qui — en Locomotive Basic — est une variable semblable à un point courant, une valeur initiale et une valeur finale. Par exemple l'instruction FOR *entries* = 0 TO 20 positionne la variable de contrôle appelée *entries* à 0 et donne la valeur à laquelle la boucle se terminera, c'est-à-dire 20. Derrière FOR, se trouvent les instructions qui doivent être répétées ; cette portion de programme devra se terminer par l'instruction NEXT *entries*. Lorsque cette ligne est exécutée, l'ordinateur remonte à l'instruction qui est située après l'instruction FOR, et la valeur des *entries* est incrémentée de 1.

Après avoir incrémenté la variable de contrôle, cette nouvelle valeur est comparée à la valeur finale. Si elle lui est supérieure, la boucle s'arrête. Il est important de bien le comprendre si l'on utilise la variable de contrôle une fois la

boucle terminée, celle-ci étant alors supérieure à la valeur finale.

Normalement, la variable de contrôle est incrémentée de 1 à chaque fois que l'instruction est exécutée, mais on peut modifier ceci grâce à l'instruction optionnelle STEP.

Par exemple FOR *Columns* = 0 TO 39 STEP 5 augmentera la variable de contrôle de 5 à chaque fois et FOR *dots* = 0 TO 2*PI STEP 0,03 augmentera sa valeur de 0,03. Ce deuxième exemple montre aussi que les valeurs initiales et finales de la variable de contrôle ne sont pas nécessairement des constantes. En fait, ces valeurs de même que la taille du pas peuvent être des constantes, des variables, ou bien des expressions.

Il est également possible de choisir un pas négatif. Dans ce cas, il est essentiel, si l'on veut que la boucle s'effectue plus d'une fois, que la valeur initiale de la variable de contrôle soit supérieure à sa valeur finale. Les boucles FOR... NEXT sont exécutées au moins une fois, même si la valeur initiale de la variable de contrôle est supérieure à sa valeur finale (dans le cas d'un pas positif). Ceci parce que la valeur de la variable de contrôle n'est pas vérifiée, tant que l'instruction NEXT n'est pas exécutée.

On peut très bien utiliser la variable de contrôle pour des calculs ou toute autre chose à l'intérieur de la boucle. C'est pour cela qu'il est très utile d'avoir un pas variable. En utilisant la variable de contrôle comme coordonnée dans une instruction LOCATE, un caractère peut être inscrit à des endroits différents de l'écran : c'est un outil très utilisé pour la décoration graphique. On peut aussi aisément, grâce à la variable de contrôle, écrire un programme qui affiche, par exemple, les racines carrées des nombres compris entre 1 et 10.

Il faut cependant veiller à ne pas modifier la valeur de la variable de contrôle à l'intérieur de la boucle. Vous ne ferez pas une erreur, car l'ordinateur s'en accommode, mais cette pratique est mauvaise et doit être évitée. Si vous ne permettez pas à la variable de contrôle d'atteindre sa valeur finale, alors la boucle s'effectuera indéfiniment. De plus,

vous ne pouvez pas sortir d'une boucle avec un GOTO, et cela peut induire l'ordinateur en erreur.

Les boucles WHILE... WEND exécutent itérativement les instructions comprises entre l'instruction WHILE et l'instruction WEND, tant que la condition introduite par WHILE est vraie (le sens exact du mot *vrai* sera étudié dans un prochain chapitre). Si la condition relative à l'instruction WHILE n'est pas vraie, lorsque l'instruction est exécutée ; alors la prochaine instruction traitée sera celle qui suivra WEND, c'est pourquoi une boucle WHILE... WEND n'est pas nécessairement exécutée entièrement. Ainsi cette boucle s'oppose aux boucles FOR... NEXT et REPEAT... UNTIL recontrées dans d'autres Basic. Il est souvent bénéfique de considérer une boucle WHILE... WEND comme une boucle à entrée conditionnelle plutôt qu'à sortie conditionnelle.

TABLEAUX

Jusqu'à présent nous avons rencontré des variables simples ou de contrôle. Il en existe une troisième catégorie. Ce sont les tableaux ou variables dimensionnées. Ce sont des groupes de variables, ayant le même nom, se différenciant des autres par leur nombre ou leur dimension.

Les tableaux utilisent beaucoup de mémoire, c'est pourquoi l'ordinateur doit leur réserver de la place. Cela s'effectue grâce à l'instruction DIM (Dimension). Ainsi DIM *subtotals (11)* créera un tableau de 12 éléments numérotés de 0 à 11 (beaucoup de Basic comptent les éléments à partir de 0, mais quelques-uns commencent à 1). Dans Locomotive Basic, il n'est pas toujours nécessaire de dimensionner (ou déclarer) un tableau. Si vous ne le faites pas, l'ordinateur imposera une longueur de 10 éléments.

Pour accéder à un élément du tableau, vous utilisez son nom et son numéro, par exemple PRINT *subtotals (7)*. Le nombre entre parenthèses n'est pas nécessairement constant, il peut être une expression ou une variable. En fait c'est souvent la variable de contrôle d'une boucle FOR... NEXT. Les boucles et les tableaux vont de pair et sont la base

de nombreuses techniques de programmation très utiles, notamment pour les problèmes de tri.

Listing 1 est un programme qui utilise à la fois des tableaux et des boucles pour résoudre un problème que l'on rencontre souvent dans les énigmes proposées par les magazines informatiques.

LISTING 1

```
2Ø DIM myshare(1ØØ)
3Ø FOR bags=1 TO 1ØØ
4Ø myshare(bags)=bags-(bags*bags/1ØØ)
5Ø NEXT bags
6Ø FOR bags=1 TO 1ØØ
7Ø IF myshare(bags)>besthare THEN
   bestshare=myshare(bags):highest=bags
8Ø NEXT bags
9Ø PRINT highest, bestshare
```

Le problème est le suivant : vous êtes un pauvre fermier. Votre insatiable propriétaire prélève sur votre récolte un pourcentage proportionnel au nombre de sacs de blé que vous avez produit. Pour un sac il vous prend 1 pour cent de votre récolte, pour 10 sacs, 10 pour cent et ainsi de suite. Quel est le nombre de sacs que vous devez produire de façon à faire le plus de profit ? Il y a sûrement une formule mathématique exprimant le résultat, mais la façon de procéder la plus facile avec un ordinateur est de trouver une réponse pour chaque nombre de sacs, de ranger ces résultats dans un tableau et de chercher dans le tableau le plus grand résultat. De toute évidence, le plus grand nombre de sacs à considérer est 100, un nombre supérieur produisant un résultat négatif.

La ligne 20 dimensionne le tableau. Les lignes 30-60 sont les éléments de la première boucle, la ligne 40 calculant les éléments du tableau. La seconde boucle comprend les lignes 60-80. La ligne 70 compare une nouvelle variable, *bestshare,*

avec le contenu de chaque élément du tableau. Si un élément du tableau lui est supérieur alors *bestshare* prend cette nouvelle valeur, et la nouvelle variable, *highest*, prend la valeur de la variable de contrôle.

Lorsque la boucle est finie, *bestshare* est égal au plus grand élément du tableau et *highest* indique la position de cet élément dans le tableau. La ligne 90 affiche la solution.

Notons que dans ce programme, la première boucle est terminée avant que la seconde ne soit commencée. Mais il est possible d'avoir une boucle qui soit à l'intérieur d'une autre ; ce procédé porte le nom de *nesting*. Le listing 2 en est une illustration graphique ; il imprime sur l'écran des étoiles disposées en rectangle.

LISTING 2

```
2Ø CLS
3Ø FOR y=2 TO 2Ø : REM bas en haut
4Ø FOR x=4 TO 36 : REM gauche à droite
5Ø LOCATE x, y
6Ø PRINT "*"
7Ø NEXT x
8Ø NEXT y
```

La ligne 20 efface l'écran. La première boucle est entre les lignes 30 et 80. La seconde va des lignes 40 à 70. Remarquons que la seconde boucle est complétement contenue dans la première. C'est une des règles des boucles : ou bien l'une est complètement à l'intérieur de l'autre, ou bien elles sont complètement séparées comme dans le Listing 1. Si elles se chevauchent, il y aura erreur.

La ligne 50 positionne le curseur, les variables de contrôle servant de coordonnées, et la ligne 60 imprime les étoiles.

Lorsque vous faites exécuter le programme, vous remarquez que chaque ligne horizontale est achevée avant que la suivante ne commence. La position horizontale est contrôlée par la boucle X.

Une fois le programme exécuté, listez-le, puis intervertissez les lignes 30 et 40 ; si vous essayez alors d'exécuter le programme, vous n'y parviendrez pas, et vous obtiendrez un message d'erreur car les boucles se chevauchent. Maintenant, intervertissez les lignes 70 et 80. Le programme fonctionnera de nouveau, mais cette fois chaque ligne verticale sera terminée avant qu'une nouvelle ne commence.

Le *nesting* peut comporter plus de 2 boucles si cela est nécessaire. De même le *nesting* est possible entre les boucles WHILE et les boucles FOR. Elles doivent alors être séparées, ou l'une à l'intérieur de l'autre.

Les tableaux rencontrés jusqu'à présent n'avaient qu'une dimension, comme une liste de nombres sur une page. Il est possible, en Locomotive Basic de créer des tableaux bidimensionnels semblables à des colonnes de nombres sur une page écrite de bas en haut et de gauche à droite.

Le listing 3 démontre l'utilité de tableaux à deux dimensions associés à des boucles. La ligne 30 dimensionne le tableau, il y a 6 éléments (numérotés de 0 à 5) dans chaque dimension. Il faut noter qu'il y a 6*6 = 36 éléments et non 12. La première boucle va des lignes 40 à 80, la seconde des lignes 60 à 70. La ligne 60 range dans le tableau le résultat de la multiplication des variables de contrôle. Le reste du programme ne sert qu'à afficher le résultat. Cette partie est à comparer avec le listing 2.

LISTING 3

```
2Ø CLS
3Ø DIM answers(5,5)
4Ø FOR factor1=Ø TO 5
5Ø FOR factor2=Ø TO 5
6Ø answers(factor1,factor2)=factor1*factor2
7Ø NEXT factor2
8Ø NEXT factor1
9Ø FOR lines=Ø TO 5
1ØØ FOR columns=Ø TO 5
11Ø LOCATE 5*columns+2,3*lines+1
```

```
12Ø PRINT answers(columns,lines)
13Ø NEXT columns
14Ø NEXT lines
```

Il est possible d'utiliser plus de deux dimensions. Un tableau trois dimensions serait semblable à un livre dont la première dimension serait les colonnes, la seconde les lignes, la troisième les pages. Le listing 4 illustre ceci. C'est une extension du Listing 3. Une dimension supplémentaire est définie à la ligne 30 ; il y a de même une boucle supplémentaire à la ligne 70. Nous ne pouvons inscrire que deux dimensions sur l'écran, c'est pourquoi la seconde partie du programme contient une instruction INPUT qui permet de choisir la page que l'on veut regarder (ligne 110). La ligne 120 est une manière d'arrêter le programme si un numéro de page non valable est demandé. La ligne 200 permet de sélectionner une nouvelle page, la précédente étant affichée.

LISTING 4

```
2Ø CLS
3Ø DIM answers(5,5,25)
4Ø FOR factor1=Ø TO 5
5Ø FOR factor2=Ø TO 5
6Ø FOR factor3=Ø TO 5
7Ø answers(factor1,factor2,factor3)=factor1*factor2*factor3
8Ø NEXT factor3
9Ø NEXT factor2
1ØØ NEXT factor1
11Ø INPUT "Quelle page (Ø à 25)",page%
12Ø IF page%<Ø OR page%>25 THEN STOP
13Ø CLS
14Ø FOR lines=Ø TO 5
15Ø FOR columns=Ø TO 5
16Ø LOCATE 5*columns+2,3*lines+1
17Ø PRINT answers(columns,lines,page%)
18Ø NEXT columns
19Ø NEXT lines
2ØØ GOTO 11Ø
```

Les tableaux multidimensionnels et le *nesting* sont très utiles et constituent une partie importante de la programmation. Il est important d'y passer le temps nécessaire, de façon à les comprendre complètement.

Chapitre 2

VARIABLES CHAÎNES

Dans le premier chapitre, nous avons étudié la façon dont l'ordinateur traitait les nombres. Mais l'ordinateur doit aussi mémoriser du texte. En programmation, on accède au texte en mémoire à l'aide de chaînes. En d'autres termes le texte est traité comme une suite ordonnée de caractères liés les uns aux autres.

Comme pour les variables numériques, il est possible de représenter une chaîne de caractères par une variable et d'imprimer cette variable. On peut également transférer le contenu d'une variable dans une autre (si cela est fait, comme pour les variables numériques, la valeur originale n'est pas détruite).

Les variables chaînes se distinguent par le dernier caractère de leur nom, qui est le signe dollar ; par exemple *name$*, *search$*. Pour assigner une valeur à une chaîne de caractères, les caractères doivent être insérés entre des guillemets inversés ou des apostrophes. Ainsi on a LET *name$="John"* ou LET *name$="Robert"*. Les chaînes de caractères entre guillemets, alternativement inversés, sont appelées lettres chaînées et sont équivalentes aux constantes numériques (appelées occasionnellement constantes chaînées).

Il est possible de réaliser toute une gamme de manipulations sur ces chaînes. La première d'entre elles est la concaténation, qui signifie simplement que les chaînes sont mises bout à bout. De nombreux Basic utilisent le signe + pour cela, mais il est important de ne pas faire de confusions avec le signe arithmétique. Par exemple, LET *a$="Hello"*:*b$="Sailor"*:c$=a$+*b$*:PRINTc$ affichera *HelloSailor*. Remarquons qu'il n'y a pas d'espace entre les mots.

De même qu'il est possible d'imprimer ou de manipuler des chaînes entières, il est également possible de caractériser un élément de la chaîne. Il y a trois fonctions de chaînes qui effectuent cela. LEFT$ est utilisée pour sélectionner les

premiers caractères de la chaîne (en partant de la gauche). RIGHT$ sélectionne les derniers caractères. MID$ peut être utilisé pour sélectionner les caractères à partir du milieu de la chaîne de gauche à droite, ou en sens inverse si nécessaire. Ces fonctions sont utilisées pour particulariser un caractère seulement, ou un nombre consécutif de caractères. Le nombre de caractères et la chaîne que l'on veut sélectionner sont mis entre crochets après le nom de la variable. Pour l'instruction MID$ il faut aussi donner la position du premier caractère pris.

Par exemple, si nous écrivons *a$="chopper"*, PRINT LEFT$(a$,4) affichera *chop,* PRINT RIGHT$ (a$,3) indiquera *per.* PRINT MID$ (a$,2,3) affichera *hop.* Dans l'instruction MID$ le point de départ est donné en premier, suivi du nombre de caractères que l'on désire prendre. On peut imprimer directement les caractères sélectionnés, mais on peut aussi les transférer ou les concaténer à d'autres chaînes. Dans tous les cas, la chaîne originale reste intacte.

Dans Locomotive Basic, la longueur des chaînes n'est pas fixée. Cette longueur s'adapte au nombre de caractères contenus dans la chaîne. L'ordinateur doit mémoriser la taille de la chaîne de façon à connaître le nombre de caractères qu'il doit imprimer (ou toute autre manipulation). Cette longueur est contrôlée par la fonction LEN(). Une chaîne a une longueur maximale de 255 caractères.

Le listing 5 exploite la plupart des idées que nous venons d'énoncer. Ce programme a pour origine une lettre de lecteur à un magazine informatique. Ce lecteur voulait un programme qui créât une personnalité, et un nom pour les personnages de son jeu d'aventure. Créer une personnalité est difficile, mais ce programme vous donnera des noms très valables.

LISTING 5

```
2Ø REM * Création de noms *
3Ø source$="CREATINGNAMESFORADVENTUREGA
MESISFUNWITHANAMSTRADCOMPUTER"
```

```
40 WHILE 1
50 name$=""
60 FOR namelength=1 TO RND(1)*4+5
70 name$=name$+MID$(source$,RND(1)*LEN
(source$)+1,1)
80 next namelength
90 PRINT name$
100 WEND
```

Il est difficile de créer des mots au hasard. Il n'est pas très fructueux de prendre aléatoirement des lettres de l'alphabet et de les assembler, en supposant qu'elles ont la même probabilité d'être sélectionnées, car dans notre langage certaines lettres apparaissent plus fréquemment que d'autres. Ce programme tient compte de cette remarque et utilise comme référence une phrase d'anglais définie en ligne 30 par source$.

Les instructions WHILE 1 ligne 40, et WEND ligne 100 créent une boucle infinie. La ligne 50 assigne à la variable *name$* une chaîne vide. La ligne 60 est le début d'une boucle FOR... NEXT. La fonction RND est utilisée pour créer des noms d'une longueur de 5 à 9 caractères. La ligne 70 utilise l'instruction MID$ pour sélectionner aléatoirement des caractères appartenant à *source$* et les concaténer à *name$*. On se sert de la fonction LEN. Son emploi facilite l'utilisation des différentes chaînes de référence (définies à la ligne 30) sans avoir à en compter les caractères. La ligne 90 imprime les noms. Le Locomotive Basic utilisé dans l'Amstrad est très rapide et les noms qui s'inscrivent sur l'écran ne peuvent être lus. Il est possible d'arrêter le programme en appuyant une fois sur la touche ESCAPE. Le programme repart si l'on appuie sur n'importe quelle autre touche. Locomotive Basic a aussi des tableaux chaînes. Ils doivent être déclarés comme les tableaux numériques. Il est également possible qu'ils soient multidimensionnels. La taille de chaque chaîne élément est variable, comme pour les chaînes ordinaires, le maximum étant toujours de 255 caractères.

Nous avons signalé au chapitre 1 que l'utilisation simulta-

née des boucles et des tableaux est la solution de nombreux problèmes de tri. Le Listing 6 illustre cela. Ce programme est divisé en quatre parties : une introduction des lignes 20 à 40, une procédure permettant d'écrire les chaînes à partir du clavier des lignes 50 à 130, puis le tri proprement dit des lignes 150 à 260, et enfin de la ligne 280 jusqu'à la fin une procédure qui imprime les chaînes triées.

LISTING 6

```
20 REM * Exemple de tri *
30 entries=-1
40 DIM store$ (20)
50 PRINT "Ecrivez les chaînes s'il vous plaît."
60 entry$="*"
70 WHILE entry$<>""
80 INPUT entry$
90 IF entry$="" THEN 130
100 LET entries=entries+1
110 store$(entries)=entry$
120 IF entries=20 THEN PRINT "Le tableau est maintenant rempli"
130 WEND
140 PRINT "L'ordinateur trie"
150 FOR strings=0 TO entries-1
160 compare$=store$(strings)
170 position=strings
180 FOR rest=strings+1 TO entries
190 IF compare$>store$(rest) THEN compare$=store$
    (rest):position=rest
200 NEXT rest
210 IF position=strings THEN 260
220 FOR moves=position-1 TO strings STEP-1
230 store$(moves+1)=store$(moves)
240 NEXT moves
250 store$(string$)=compare$
260 NEXT strings
270 PRINT "...terminé"
```

```
28Ø FOR string$=0 TO entries
29Ø PRINTstore$(strings)
3ØØ NEXT strings
```

La ligne 40 indique que le tableau (une seule dimension) peut contenir 21 chaînes. La variable *entries* est utilisée pour les compter, elle est initialisée à la valeur − 1.

Une boucle WHILE ... WEND sert à entrer les chaînes, cette opération étant effectuée à la ligne 80. La ligne 90 associée à la condition WHILE autorise la fin de la procédure d'entrée des chaînes sans avoir à déclarer les 21 permises : il suffit d'appuyer sur la touche *enter*, et de ne rien écrire. Lorsque l'on rentre une chaîne, *entries* est incrémentée de 1, et la chaîne se trouve alors dans le tableau à la position correspondant à la valeur d'*entries*. C'est pourquoi la valeur initiale d'*entries* est − 1, la première chaîne se trouvant être l'élément 0. La ligne 120 indique que le tableau est rempli.

La méthode de tri intervient alors. Grâce à une boucle FOR (ligne 180) le contenu du premier élément du tableau (sélectionné par la variable de contrôle de chaîne en ligne 160) est reproduit dans la variable *compare$*. On donne alors à la variable *position* la valeur 0 (ligne 170). La variable *compare$* est ensuite comparée tour à tour à tous les éléments du tableau. Si le contenu de l'un des éléments du tableau est antérieur dans l'ordre alphabétique à la valeur de *compare$*, « position » prend la valeur de la position de cet élément (ligne 190). Après une première série de comparaisons, la première des chaînes (ordre alphabétique) se trouve dans *compare$* et sa position est donnée par la variable position ; si cette chaîne est déjà en première position dans le tableau, il n'y a rien à faire, cela se traduit par la ligne 210.

Dans le cas contraire, il faut utiliser la partie du programme qui place cette chaîne en première place. Les lignes 220 à 240 déplacent d'une position toutes les chaînes situées au-dessus de celle à placer en tête du tableau. La valeur de *compare$* est alors rangée dans l'élément zéro du tableau (ligne 250).

Lors de la seconde réalisation de la boucle FOR, le

deuxième élément du tableau est comparé à toutes les chaînes suivantes et, si cela est nécessaire, la seconde de l'ordre alphabétique est placée en seconde place. Cette procédure se répète jusqu'à ce que l'avant-dernier élément soit comparé au dernier.

Les chaînes sont alors en ordre et les lignes 280 à 300 sont imprimées. Cette méthode de tri est considérablement plus rapide qu'un tri classique et aléatoire. Elle a aussi l'avantage de ne pas changer l'ordre des éléments si ce n'est pas nécessaire, ce qui convient aux tris à champs multiples. Si vos données sont des textes datés et si vous triez ces données par ordre alphabétique puis en fonction de la date, les textes ayant la même date resteront dans l'ordre alphabétique.

Il est intéressant de savoir comment un ordinateur peut mémoriser des lettres. En fait chaque caractère est codé par un nombre, ce sont ces nombres qui sont pris en compte. Il existe deux fonctions qui donnent le nombre codé de chaque caractère et inversement le caractère à partir du nombre codé. La première d'entre elle s'appelle ASC(). Ces initiales correspondent à *American Standard Code of Information Interchange* (ASCII) : c'est le code interne de l'Amstrad ainsi que de la plupart des ordinateurs. Ainsi PRINT ASC("A") imprimera 65 qui est le nombre-code correspondant à A. Cette fonction s'utilise aussi pour les variables chaînes. Si une chaîne comporte plus d'un caractère, le code correspondant sera celui du premier élément de la chaîne. Il est bien entendu possible de sélectionner des caractères placés au milieu de la chaîne, par exemple PRINT ASC(RIGHT$(a$,3)). Si la sélection comporte plusieurs caractères, le code correspondra uniquement au premier.

LISTING 7

```
20 REM code ASCII
30 WHILE 1
40 a$=INKEY$:IF a$="" THEN 40
50 PRINT ASC(a$)
60 WEND
```

Le Listing 7 est un programme qui imprime le code ASCII de la touche sur laquelle on appuie. Certaines instructions, comme CTRL ou SHIFT, ne génèrent pas elles-mêmes de nombres codés, mais elles modifient le codage des touches enfoncées simultanément.

La fonction CHR$() donne, sous forme de chaîne, le caractère correspondant au code ASCII indiqué entre parenthèses. PRINT CHR$(65) inscrira A sur l'écran. Cette fonction est utile si l'on veut imprimer des caractères graphiques qui ne sont pas disponibles sur le clavier, ainsi que pour imprimer certains éléments de contrôle ; par exemple CHR$(10) détruit les lignes et CHR$(12) équivaut à CLS.

Le listing 8 imprime la signification des nombres de code que vous inscrivez. Le nombre le plus élevé possible est 255. Les nombres inférieurs à 32 sont les instructions de contrôle, et certaines d'entre elles ont des effets surprenants.

LISTING 8

```
2Ø REM caractères
3Ø WHILE 1
4Ø INPUT "code du nombre";code
5Ø PRINT CHR$(code)
6Ø WEND
```

Il existe deux fonctions de chaîne que seuls quelques ordinateurs possèdent, dont l'Amstrad. Ce sont UPPER$ et LOWER$. Elles ont pour effet de convertir tous les caractères d'une chaîne en majuscules ou en minuscules. Elles sont utilisées pour le tri, et la recherche des chaînes. Des exemples d'utilisation seront données dans les chapitres suivants.

Chapitre 3

LES DÉCISIONS

Lors de décisions, l'instruction IF ... THEN est appropriée. Elle a la forme suivante : IF *condition* THEN *instruction*. Si la *condition* est vraie, *l'instruction* est exécutée, sinon le programme traite la ligne suivante. Cependant IF ... THEN possède une ou deux astuces utiles à la programmation, mais qui peuvent être déroutantes si l'on n'y fait pas attention.

Il est possible d'écrire d'autres instructions sur la même ligne que IF ... THEN, mais il faut se souvenir que si la condition introduite dans IF ... THEN est fausse, alors le programme exécutera la ligne suivante. Cela signifie qu'aucune des instructions de la ligne ne sera éxécutée. Cela peut être très utile lorsque vous désirez que plusieurs instructions se réalisent si la condition est vraie. ELSE est une extension optionelle de l'instruction IF ... THEN. Lorsque l'on utilise ELSE, si la condition qui précède IF est vraie, toutes les instructions comprises entre THEN et ELSE sont exécutées. ELSE pose plus de problèmes qu'elle n'en résout, et s'utilise rarement.

Vous prenez moins de place mémoire en écrivant plusieurs instructions sur une même ligne qu'en les écrivant sur plusieurs. Si le programme prend toute la place mémoire, il est souvent bénéfique de rassembler deux (ou plus) lignes en une seule. Si vous faites cela, n'oubliez pas de mettre les instructions supplémentaires après un IF ... THEN.

L'instruction IF ... THEN agit différemment suivant que la condition est vraie ou fausse. Il est intéressant de comprendre comment cette décision est prise. Essayez de taper le programme suivant :

```
A=6
PRINT A=6
PRINT 6=6
PRINT A=5
PRINT 6=5
PRINT A=B
```

Vous vous attendez à un message d'erreur pour chaque ligne. En fait les deux premières donneront −1 et les trois dernières 0. Ceci parce que l'ordinateur évalue ces lignes comme des expressions vraies ou fausses, or, −1 correspond à vrai, et 0 à faux.

De toute évidence, A vaut 6. C'est la valeur que nous lui avons attribuée. De même 6 égal 6, A n'est pas égal à 5 et 6 est différent de 5. Dans le dernier exemple A est différent de B, puisque A est égal à 6 et que l'on n'a pas affecté de valeur à B, que l'on considère donc comme nulle.

Ces valeurs correspondant à *vrai* et *faux* restent valables pour les comparaisons de chaînes ainsi que pour les expressions *plus que, moins que, différent de,* etc.

Grâce à cela, il est possible de tester une variable seule à l'aide d'une instruction IF ... THEN. Essayez le programme suivant :

```
IF A THEN PRINT "VRAI"
IF B THEN PRINT "VRAI"
IF NOT A THEN PRINT "FAUX"
```

La première ligne donnera VRAI puisque chaque valeur non nulle peut être considérée comme vraie. La seconde ligne ne donnera rien puisque B contient 0, donc la proposition est fausse.

Le troisième exemple est plus complexe. NOT inverse le résultat d'un test, donc FAUX sera imprimé si A est faux. Dans notre exemple, c'est le cas. Parce que l'on utilise NOT, seul −1 représente une vraie valeur. Tout autre valeur est considérée comme fausse. D'autres ordinateurs comme Atari ou Sinclair utilisent +1 plutôt que −1 pour représenter vrai ; dans ce cas, NOT déclarera vraie toute valeur non nulle. Il faut y faire attention lorsque l'on traduit sur l'Amstrad des programmes écrit pour ces ordinateurs.

Ces tests sur une seule variable ne sont pas purement académiques. Ils permettent d'économiser de la mémoire et surtout, bien souvent, du temps.

AND ET OR

Dans Locomotive Basic, AND et OR peuvent être utilisés comme opérateurs logiques ; ils servent à tester deux conditions-ou plus-lors d'une instruction.

AND s'utilise essentiellement comme le mot "et" en français. Si les conditions introduites par IF et reliées par AND sont vraies, alors l'instruction suivant THEN sera exécutée. Si elles sont fausses, alors cette instruction ne sera pas traitée. OR s'emploie de la même manière. Si l'une des conditions introduites par IF et reliée par OR est vraie, alors l'instruction qui suit THEN est exécutée. Cet emploi est largement différent de celui du mot "où" en français.

L'emploi simultané de AND et OR n'est pas toujours évident. En voici une illustration :

Soit un programme avec trois variables A, B, C. Certains événements ont lieu si A=6 et B=3 ou C=7 et B=3. On ne peut pas écrire le programme ainsi :

```
IF A=6 AND B=3 OR C=7 AND B=3 THEN...
```

L'ordinateur n'associera pas les deux conditions à droite de OR ainsi que celles de gauche, contrairement à l'emploi de "ou" dans notre langue. Si C=7, les instructions après THEN sont toujours exécutées. De même si A=6 et B=3, ce qui n'est pas désirable. Une manière de s'en sortir est d'utiliser des parenthèses :

```
IF (A=6 AND B=3) OR (C=7 AND B=3) THEN ...
```

L'ordinateur évalue d'abord les quantités entre parenthèses. Si l'une d'elles est vraie, alors les instructions qui suivent THEN sont exécutées.

On aurait aussi pu écrire :

```
IF B=3 AND (A=6 OR C=7) THEN ...
```

C'est même préférable puisque l'expression est plus courte et sera traitée plus rapidement.

Si vous utilisez NOT lors d'une condition, souvenez-vous qu'il n'affecte que l'expression immédiatement derrière. Il n'opère plus sur une expression entière. Par exemple :

IF NOT A=6 AND B=3 THEN...

ne traitera l'instruction qui suit THEN que si B=3 et A égal à toute valeur autre que 6.

ON

ON permet une alternative lors de décisions. On l'emploie lorsqu'il faut choisir un élément parmi une liste de possibilités. ON n'est utilisée qu'avec deux instructions : GOSUB ou GOTO.

L'instruction ON a la forme suivante : ON *variable GOSUB (ou GOTO) numéro de ligne, numéro de ligne, numéro de ligne...*

La valeur de la variable détermine à quel numéro de ligne l'ordinateur ira travailler. Si elle vaut 1 l'ordinateur ira en première ligne. Si elle vaut 2 il ira en seconde ligne et ainsi de suite. Si la valeur de la variable est zéro, ou est supérieur au nombre de lignes, le programme exécute l'instruction suivante.

ON ... GOSUB s'emploie généralement avec des listes de données. Une liste comme celle-ci est inscrite sur l'écran :

1. Entrées des données
2. Chargement de la file
3. Edition de la file
4. Recherche à l'intérieur de la file
5. Tri de la file
6. Rangement de la file

   Choisissez, s'il vous plaît.

L'utilisateur tape le numéro de code choisi, la variable prend cette valeur, elle est ensuite utilisée dans l'instruction ON ... GOSUB pour appeler la fonction que l'on veut utiliser. L'instruction ON ... GOSUB devrait normalement être suivie d'un GOTO pour que l'ordinateur revienne au début de la liste de façon à ce qu'elle réapparaisse une fois l'opération

terminée, ou bien si le numéro sélectionné est supérieur au nombre d'options de la liste.

Bien que ce cas d'utilisation de ON soit le plus fréquent, ce n'est pas le seul. ON est une instruction utile et jamais assez employée. Lorsqu'un choix doit être fait, il faut toujours penser à ON qui est bien plus rapide qu'une longue liste de IF ... THEN.

Le mot-clé ON a plusieurs emplois en Locomotive Basic, par exemple ON ERROR, ON BREAK. Il n'y a pas de relations particulières avec ON utilisé lors d'un choix.

Le listing 9 est un jeu appelé : *"Devinez le nombre"*. Il utilise un grand nombre d'instructions conditionnelles. Le principe de ce jeu est le suivant : vous créez tout d'abord un nombre au hasard ; on vous donne alors une indication quant à sa taille. Puis vous énoncez vos suggestions et l'ordinateur vous indique la marche à suivre. Des REM*arques* sont incluses dans le programme.

Essayez de suivre le déroulement du programme à l'aide du listing. C'est une manière bénéfique de comprendre comment le programme est traité.

LISTING 9

```
2Ø REM * Devinez le nombre *
3Ø MODE 1:tries=1:oldiff=5ØØØ
4Ø CLS
5Ø LOCATE 5,5:PRINT "Je pense à un nombre..."
6Ø LOCATE 5,7:PRINT "... il est compris entre 1 et 5ØØØ."
7Ø REM cette boucle crée un nombre au hasard
8Ø LOCATE 5,22:PRINT "APPUYER SUR UNE TOU-
CHE POUR JOUER"
9Ø WHILE INKEY$=""
1ØØ number=INT(RND(1)*5ØØØ)
11Ø WEND
12Ø LOCATE 5,22:PRINT SPACE$(19)
13Ø LOCATE 5,10:PRINT "Voici un indice..."
14Ø LOCATE 5,15:PRINT "ce pourrait être..."
15Ø LOCATE 5,17
```

```
16Ø IF number<1ØØ THEN PRINT "l'âge d'une personne."
17Ø IF number>99 AND number<4ØØ THEN PRINT "le score d'un match de cricket."
18Ø IF number>399 AND number<1ØØØ THEN PRINT "le nombre de page d'un livre."
19Ø IF number>999 AND number<3ØØØ THEN PRINT "la cylindrée d'une voiture."
2ØØ IF number>2999 THEN PRINT "un numéro de téléphone."
21Ø LOCATE 5,19:PRINT "Essayer un numéro";tries
22Ø LOCATE 18,19:INPUT "quel est votre choix ?";guess
23Ø guess=INT(guess)
24Ø IF guess<1 OR guess>5ØØØ THEN LOCATE 5,19:PRINT "IMBECILE !!!":GOTO 22Ø
25Ø diff=ABS(number-guess)
26Ø REM passe en fin de programme si le numéro choisi est bon
27Ø IF diff=Ø THEN 41Ø
28Ø CLS
29Ø REM détermine la taille de l'erreur et donne un indice
3ØØ LOCATE 5,5
31Ø IF number<guess THEN PRINT "trop grand";
32Ø IF number>guess THEN PRINT "trop petit";
33Ø IF diff<1Ø THEN PRINT "Mais vous êtes tout près.";
34Ø IF diff>9 AND diff<5Ø THEN PRINT "Mais vous n'êtes pas loin.";
35Ø IF diff>49 AND diff<2ØØ THEN PRINT "."
36Ø IF diff>199 THEN PRINT "Vous en êtes loin !"
37Ø IF diff>oldiff THEN LOCATE 5,10 :PRINT "Vous êtes plus éloigné que la dernière fois !"
38Ø tries=tries+1:oldiff=diff
39Ø LOCATE 5,15:PRINT "Dernier choix"; guess
4ØØ GOTO 21Ø
41Ø CLS
42Ø x=5
43Ø FOR y=5 TO 2Ø
44Ø LOCATE x,y:PRINT "BRAVO !!!"
45Ø x=x+1
46Ø NEXT y
```

```
47Ø LOCATE 5,22:PRINT "Voulez-vous continuer (oui(y)/
non(n)) ?"
48Ø ans$=INKEY$:IF ans$=""THEN 48Ø
49Ø IF LOWER$(ans$)="y" THEN RUN
5ØØ END
```

Chapitre 4

INPUT, PRINT, DATA

La plupart des programmes usuels nécessitent, lors de l'exécution, des moyens de communication avec l'utilisateur. L'instruction INPUT provoque un arrêt du programme pendant lequel les données sont communiquées.

Cette interruption est signalée par un point d'interrogation sur l'écran. La taille de la donnée n'étant pas restrictive, l'ordinateur doit savoir lorsque l'utilisateur a fini d'écrire. La donnée est considérée comme terminée lorsque l'on appuie sur la touche RETURN, l'exécution du programme reprend alors.

INPUT est utilisé pour recevoir des données chaînées ou numériques. Une fois entrée, la donnée est placée dans une variable. La nature de cette variable, déterminée par l'instruction INPUT, précise si l'on a affaire à une chaîne ou à une donnée numérique. Cependant une chaîne peut très bien contenir des nombres. Par contre si un caractère qui n'est pas numérique est introduit par une variable numérique, il y a erreur : le message *? Redo from Start* est alors imprimé.

Il est possible d'introduire une donnée dans plusieurs variables, grâce à l'instruction INPUT ; pour cela, il suffit d'écrire les variables, après INPUT, séparée par des virgules ; voici un exemple :

```
100 INPUT NUM1,NUM2,NUM3
250 INPUT NAME$,ADDR$
```

Lorsque la donnée correspond à un ensemble de questions, chaque réponse doit être séparée par une virgule, il faut alors appuyer sur RETURN lorsque la donnée est complètement introduite. Ceci est valable pour les chaînes et les données numériques. Il est donc nécessaire de ne pas mettre de virgules dans les chaînes répondant à l'instruction INPUT. Pour résoudre ce problème, il y a l'instruction LINE INPUT.

Elle prendra tous les caractères inscrits, virgules et virgules inversées comprises, mais elle n'est utilisable que pour une seule variable à la fois.

Il est permis de mélanger des chaînes et des caractères numériques dans une seule instruction.

C'est un autre problème de savoir si ce mélange est judicieux. Souvent des problèmes surviennent lorsqu'on inscrit d'une seule traite la réponse à plusieurs INPUT ; ceci doit donc être utilisé avec parcimonie.

L'instruction INPUT pose un problème : le point d'interrogation inscrit sur l'écran ne donne pas d'indication quant à la donnée demandée. Pour résoudre ce problème, Locomotive Basic peut insérer une ligne de texte dans l'instruction INPUT ; cette ligne est inscrite avec le point d'interrogation. Lorsque l'on utilise cette possibilité, si on laisse un point-virgule entre la phrase et le nom de la variable, alors, le point d'interrogation apparaît. Sinon il est supprimé.

Les programmeurs amateurs adorent écrire des programmes distrayants et faciles d'accès ; l'emploi des lignes de texte définies ci-dessus occupent alors une place très importante. Elles doivent indiquer clairement ce que l'on demande à l'utilisateur. Si NOM ? apparaît sur l'écran, vous comprendrez très bien le sens de la question, mais il est plus poli d'écrire *"indiquez votre nom s'il vous plaît"* ; dans ce cas, le point d'interrogation est supprimé, sinon *"Quel est votre nom s'il vous plaît ?"*, serait la meilleure possibilité.

Il faut toutefois éviter d'être pompeux : à la question *"Auriez-vous l'amabilité de donner votre nom, s'il vous plaît ?"*, il est difficile de résister à l'envie de répondre NON !

Il est très difficile de créer des phrases explicitant des instructions INPUT multiples, si ce n'est les très simples *"Donnez trois nombres"*. Dans l'ensemble, lorsque l'on écrit des programmes destinés à d'autres utilisateurs (et surtout pour des non-programmeurs), il est préférable de séparer les instructions INPUT par une ligne explicative.

PRINT

PRINT est l'instruction la plus générale qui soit pour inscrire du texte, des données numériques, des graphiques sur l'écran ; elle doit donc être à la fois diversifiée et flexible.

PRINT est suivie d'une liste d'éléments qui peuvent être des chaînes, des chiffres, des variables, des constantes ou bien n'importe quelle permutation des éléments ci-dessus ; il peut aussi y avoir des calculs, des manipulations de chaînes ou des comparaisons logiques.

Les tableaux numériques ou chaînes peuvent aussi apparaître grâce à des instructions PRINT.

Certains signes de ponctuation apparaissent dans la liste d'éléments pour contrôler et ordonner l'inscription de ces éléments sur l'écran.

Le langage Basic est né dans les années 60. A cette époque, l'interface classique avec l'ordinateur était constituée d'un clavier et d'une imprimante, plutôt que d'un VDU. Ceci explique l'emploi du mot PRINT ("imprimer" en français) et la façon dont l'instruction PRINT est exécutée.

Lorsqu'une instruction PRINT est achevée, l'ordinateur traite alors une nouvelle ligne. Si chaque élément à inscrire est écrit dans une instruction PRINT appropriée, alors, il sera inscrit seul sur une ligne.

S'il y a plusieurs éléments dans une instruction PRINT, la ponctuation utilisée indiquera comment ils seront inscrits. S'ils sont séparés par des virgules, l'ordinateur les espacera par un nombre de blancs. S'ils sont séparés par des points-virgules, alors ils seront inscrits à la file sans espaces.

Une autre ponctuation importante dans l'instruction PRINT est l'emploi de virgules inversées. Elles sont utilisées pour séparer des constantes chaînes. Elles ne sont pas imprimées ; cela signifie que l'on ne peut pas utiliser des virgules inversées à l'intérieur de constantes chaînes à inscrire.

Il n'est pas nécessaire de séparer les constantes numériques de la même manière. L'instruction PRINT 18 inscrira simplement 18. Par contre PRINT HELLO inscrira 0 car

HELLO sera considéré comme nom de variable. Le résultat de PRINT "HELLO" sera HELLO.

Voici un exemple caractéristique : 1000 PRINT *"Vous avez marqué"* ; SCORE ; *"grâce à"* ; tries ; *"tentatives"*.

Le premier élément est une constante chaîne. Il faut noter le blanc en fin de chaîne. Le point-virgule permet d'inscrire la variable numérique SCORE immédiatement après la première chaîne. Derrière SCORE il y a une autre constante chaîne, puis une variable numérique et enfin une constante chaîne. Lorsque l'instruction est exécutée, une phrase identique à celle-ci apparaîtra sur l'écran :

Vous avez marqué 18 points grâce à 7 tentatives. Cette possibilité de réunir plusieurs éléments en un seul sur l'écran est très intéressante et permet d'inscrire les résultats du programme d'une manière intelligible.

Le résultat d'une virgule dans l'instruction PRINT est la création d'une plage disponible. L'écran est considéré en un ensemble de champs de 13 colonnes de largeur. Lorsqu'une virgule (qui n'appartient pas à une chaîne insérée entre virgules inversées) apparaît sur une liste PRINT, alors le lieu d'inscription se déplace au début du champ suivant ; parfois cela coïncide avec une nouvelle ligne.

DATA, READ et RESTORE

De même qu'une variable de donnée peut être introduite pendant l'exécution d'un programme, de nombreux programmes utilisent des données de type constante. De telles données sont communiquées grâce à l'instruction DATA, elles sont placées dans des variables grâce à l'instruction READ.

L'instruction DATA est suivie d'une liste de constantes, séparées par des virgules. Cette liste peut prendre plusieurs lignes si cela est nécessaire (chacune commençant par DATA) ; ces lignes ne sont pas nécessairement groupées dans le programme. De toute façon l'ordinateur traite toutes les instructions DATA comme si elles appartenaient à une seule longue liste.

Les instructions DATA peuvent contenir à la fois des chaînes et des constantes numériques ; il n'est pas nécessaire d'insérer les chaînes entre des virgules inversées. Cependant il faut le faire si la chaîne contient une virgule.

L'instruction READ est utilisée pour ranger les constantes dans des variables. L'instruction READ traite d'abord le premier élément de la liste. L'instruction READ A$ place le premier élément de la liste de données dans la variable chaîne A$.

L'instruction READ A prendra l'élément suivant de la liste et le rangera dans la variable numérique A, que l'on utilise si l'on traite des nombres (ou tout caractère numérique, comme le point décimal par exemple). L'instruction READ peut ranger n'importe quel caractère dans une variable chaîne, mais seulement les caractères numériques dans les variables numériques.

Les éléments introduits dans les instructions DATA peuvent être lus dans des tableaux d'éléments.

Si vous essayez de *lire* (READ en anglais) plus d'éléments qu'il en existe dans la liste, il y a une erreur.

De façon à utiliser des données plusieurs fois, à l'intérieur d'un programme, on utilise l'instruction RESTORE. Elle initialise le pointeur de donnée qui se déplace pendant l'exécution du programme.

Dans Locomotive Basic, l'instruction RESTORE a un double emploi. Elle est aussi utilisée pour placer le pointeur de donnée sur une ligne particulière du progamme, de telle sorte que l'instruction READ commence en ce lieu. Cela permet des sauts à l'intérieur de la liste de donnée, et c'est une méthode très utile.

L'instruction DATA est très adaptée dans des cas tels que : la création de sons musicaux, ceci par l'intermédiaire de l'instruction SOUND.

On utilise une instruction FOR ... NEXT qui boucle un nombre de fois égal au nombre de notes ; à l'intérieur de cette boucle, il y a des instructions READ et SOUND. RESTORE placé avant FOR et suivi d'un numéro de ligne assure le bon ordre de lecture des données.

PROGRAMME DE CONVERSION MÉTRIQUE

Ce programme, Listing 10, illustre les divers points de ce chapitre et, en plus, introduit un concept important en programmation : le sous-programme (en anglais *subroutine*).

Le programme réalise des conversions empiriques/métriques et métriques/empiriques pour la plupart des unités de largeur, de poids, de surface et de volume. Ce programme est, on le conçoit aisément, assez complexe.

Le programme doit effectuer plusieurs opérations :

1. Obtenir des renseignements de l'utilisateur quant aux unités à convertir.
2. Vérifier si ces unités peuvent être traitées par le programme.
3. Si elles ne le peuvent pas, délivrer un message d'erreur.
4. Sinon obtenir des renseignements sur la quantité à convertir.
5. Calculer et imprimer le résultat.

Chacune de ces opérations est un programme en soi, c'est l'idée de base des sous-programmes. Chaque opération est réalisée comme un programme indépendant. Ils sont appelés par l'instruction GOSUB. Chaque sous-programme doit contenir une instruction RETURN ; lorsqu'elle est exécutée le programme traite l'instruction située immédiatement derrière GOSUB.

Il est possible d'appeler un sous-programme à l'intérieur d'un autre : ce procédé porte le nom de *nesting* (comme pour les boucles). Le degré de *nesting* est limité car l'ordinateur doit se souvenir de l'adresse de départ de chaque sous-programme, et l'emplacement mémoire pour cette opération est restreint.

En plus des sous-programmes définis dans la liste ci-dessus, il faut en ajouter deux. Un pour introduire le programme sur l'écran, un autre pour imprimer la liste des unités traitées par le programme. La ligne 20 est le nom du programme. La ligne 30 s'assure du bon mode d'utilisation

de l'ordinateur et efface l'écran. La ligne 40 dimensionne les tableaux. La ligne 50 appelle le sous-programme qui inscrit sur l'écran l'introduction. Ce programme est défini des lignes 1000 et 1080. La ligne 1010 imprime le titre du programme. La ligne 1020 comporte deux PRINT seuls. C'est la manière la plus simple de créer deux lignes blanches. Les lignes 1030 à 1060 créent une boucle FOR/NEXT. Elle lit (READ) les lignes à inscrire à partir des instructions DATA, puis les imprime. Les instructions DATA vont des lignes 100 à 150. Après la ligne 1080, le programme reprend en ligne 60. Le sous-programme qui est écrit des lignes 2000 à 2060 lit (READ) les unités et les facteurs de conversions dans deux tableaux. C'est plus pratique que de les utiliser directement à partir des instructions DATA. Cependant le programme aurait pu être conçu ainsi.

Le sous-programme des lignes 3000 à 3090 permet d'introduire les unités. Comme la phrase explicative est longue, elle est séparée en deux instructions PRINT, plutôt qu'en une seule. Ceci, surtout pour éviter que les mots soient coupés en bout d'écran.

Si l'utilisateur inscrit HELP, le sous-programme en ligne 4000 est appelé ; il imprime la liste des unités contenues dans le tableau.

Le programme s'arrête si l'on tape QUIT. Il faut noter ici l'emploi de UPPER$ de façon à utiliser des caractères minuscules ou majuscules.

Si l'utilisateur n'écrit pas HELP ou QUIT, le sous-programme en ligne 5000 est appelé pour aller chercher les données du tableau pour la conversion. La ligne 5010 positionne la variable *foundflag* à 0. La recherche est faite par une boucle FOR ... NEXT des lignes 5020 à 5050. Les lignes 5030 à 5040 vérifient si la conversion est possible ; si elle l'est, le facteur de conversion est rangé dans la variable *confactor* et *foundflag* est positionné à −1.

Lorsque le programme revient en 3070, si la conversion est impossible, alors *foundflag* vaut 0. Dans ce cas, le sous-programme débutant en 8000 est appelé, et inscrit un message, on passe alors en 3010 pour un nouvel essai de conversion.

La ligne 3080 est exécutée seulement lorsque la conversion est possible. La quantité est déclarée en 6050, le sous-programme à partir de la ligne 7000 est alors appelé. Notons la manière dont la conversion est réalisée dans l'instruction PRINT en 7020. Cette instruction est complexe, puisqu'elle réunit une variable numérique, une constante chaîne (un espace), une variable chaîne, une constante chaîne (le signe=), le calcul, une constante chaîne (un autre espace), et finalement une variable chaîne, toutes séparées par des points-virgules de façon à les inscrire sur une même ligne.

Une fois la conversion réalisée le programme retourne en 6030 de façon à réaliser d'autres conversions sans avoir à réintroduire les unités. Il faut toujours réaliser un programme pour qu'il soit le plus pratique possible.

Si l'on écrit 0 en 6050 l'instruction RETURN en ligne 6060 est exécutée, elle ramène le programme en ligne 80, ce qui efface l'écran. On arrive alors à la ligne 70 où il est possible de changer les unités si nécessaire.

L'instruction END à la ligne 90 ne sert que comme point de repère quand on lit le listing, elle n'est jamais exécutée. Il est utile de mettre des instructions END entre le programme principal et les sous-programmes, même si elles ne sont pas exécutées. Si l'ordinateur rencontre un RETURN sans avoir exécuté précédemment un GOSUB, il y a erreur.

La place des instructions DATA dans le programme est arbitraire. Il est judicieux de les placer entre le programme et les sous-programmes. Cependant quand elles ne sont utilisées que par un seul sous-programme, il est bon de les placer à l'intérieur de ce sous-programme.

La notion de sous-programme est essentielle, et il faut consacrer le temps nécessaire à leur compréhension. Essayer encore de suivre le déroulement de ce programme.

LISTING 10

```
2Ø REM programme de conversion métrique
3Ø MODE 1
4Ø DIM factors(9,1),unit$(9,1)
```

```
5Ø GOSUB 1ØØØ:REM inscription des instructions
6Ø GOSUB 2ØØØ:REM remplissage des tableaux
7Ø GOSUB 3ØØØ:REM entrée des données
8Ø CLS:GOTO 7Ø
9Ø END
1ØØ DATA Ce programme permet des conversions métriques/empiriques
11Ø DATA et des conversions empiriques/métriques
12Ø DATA taper "HELP" pour voir
13Ø DATA la liste des unités possibles
14Ø DATA Vous devez écrire les unités exactement comme
15Ø DATA elles apparaissent sur la liste
16Ø DATA INCHES,2.54,CENTIMETRES,.3937
17Ø DATA FEET,.3Ø48,METRES,3.281
18Ø DATA MILES,1.6Ø9,KILOMETRES,.6214
19Ø DATA SQ INCHES,6.452,SQ CENTIMETRES,.155Ø
2ØØ DATA SQ FEET,.Ø929,SQ METRES,10.76
21Ø DATA SQ MILES,2.59,SQ KILOMETRES,Ø.3861
22Ø DATA CUB INCHES,16.39,CUB CENTIMETRES,.061Ø2
23Ø DATA CUB FEET,.Ø2832,CUB METRES,35.31
24Ø DATA GALLONS, 4.546,LITRES,.22
25Ø DATA OUNCES,28.35,GRAMS,.Ø3527
1ØØØ REM instructions
1Ø1Ø LOCATE 12,5:PRINT "CONVERSION METRIQUE"
1Ø2Ø PRINT:PRINT
1Ø3Ø FOR lines=1 TO 6
1Ø4Ø READ text$
1Ø5Ø PRINT text$
1Ø6Ø NEXT lines
1Ø7Ø PRINT
1Ø8Ø RETURN
2ØØØ REM procédure de remplissage du tableau
2Ø1Ø FOR conversions=Ø TO 9
2Ø2Ø FOR systems=Ø TO 1
2Ø3Ø READ unit$(conversions,systems),factors(conversions,systems)
2Ø4Ø NEXT systems
```

```
2Ø5Ø NEXT conversions
2Ø6Ø RETURN
3ØØØ REM Procédure d'entrée des unités
3Ø1Ø PRINT "Ecrivez les unités à convertir s'il vous plaît"
3Ø2Ø PRINT "ou HELP pour voir la liste, ou QUIT."
3Ø3Ø INPUT unit$
3Ø4Ø IF UPPER$(unit$)="QUIT" THEN CLS:STOP
3Ø5Ø IF UPPER$(unit$)="HELP" THEN GOSUB
4ØØØ:GOTO 3Ø1Ø
3Ø6Ø GOSUB 5ØØØ
3Ø7Ø IF NOT foundflag THEN GOSUB 8ØØØ:GOTO 3Ø1Ø
3Ø8Ø IF foundflag THEN GOSUB 6ØØØ
3Ø9Ø RETURN
4ØØØ REM Procédure d'appel
4Ø1Ø CLS
4Ø2Ø LOCATE 12,1:PRINT "UNITES VALABLES"
4Ø3Ø FOR lines=Ø TO 9
4Ø4Ø LOCATE 1,lines*2+3:PRINT unit$(lines,Ø)
4Ø5Ø LOCATE 2Ø,lines*2+3:PRINT unit$(lines,1)
4Ø6Ø NEXT lines
4Ø7Ø LOCATE 5,24:PRINT "APPUYER SUR UNE
TOUCHE POUR CONTINUER"
4Ø8Ø ans$=INKEY$:IF ans$="" THEN 4Ø8Ø
4Ø9Ø CLS
41ØØ RETURN
5ØØØ REM Procédure de recherche
5Ø1Ø foundflag=Ø
5Ø2Ø FOR possibles=Ø TO 9
5Ø3Ø IF UPPER$(unit$)=unit$(possibles,Ø) THEN confac-
tor=factors(possibles,Ø):conv$=unit$(possibles,1):found-
flag=-1
5Ø4Ø IF UPPER$(unit$)=unit$(possibles,1) THEN confac-
tor=factors (possibles,1) :conv$=unit$ (possibles,Ø) :found-
flag=-1
5Ø5Ø NEXT possibles
5Ø6Ø RETURN
6ØØØ REM quantité introduite
6Ø1Ø PRINT
6Ø2Ø PRINT "convertir"; UPPER$(unit$);" to ";conv$
```

```
6Ø25 PRINT
6Ø3Ø PRINT "inscrivez la quantité à traduire s'il vous plaît"
6Ø4Ø PRINT "ou, Ø pour changer d'unités"
6Ø5Ø INPUT qty
6Ø6Ø IF qty=Ø THEN RETURN
6Ø7Ø GOSUB 7ØØØ
6Ø8Ø GOTO 6Ø3Ø
7ØØØ REM calcul/inscription
7Ø1Ø PRINT
7Ø2Ø PRINT qty;" ";UPPER$(unit$);"=";qty*confactor;"
";conv$
7Ø3Ø PRINT
7Ø4Ø RETURN
8ØØØ REM message d'impossibilité
8Ø1Ø PRINT
8Ø2Ø PRINT "Les unités écrites ne sont pas"
8Ø3Ø PRINT "dans la liste. Ecrivez HELP pour voir"
8Ø4Ø PRINT "la liste complète."
8Ø5Ø PRINT
8Ø6Ø RETURN
```

Chapitre 5

LE GÉNÉRATEUR DE SON

Le générateur de son du CPC 464 est très performant, puisqu'il peut simultanément créer trois sons, plus un bruitage. Ceci n'est pas exceptionnel puisque de nombreux micro-ordinateurs en font autant. Ce qui distingue le générateur de son du CPC 464 des autres, c'est son adaptation parfaite au Basic évolué, ce qui permet relativement facilement d'en tirer le maximum. Le mot *relativement* a ici une connotation importante puisque beaucoup d'utilisateurs considèrent l'utilisation du son comme un des aspects les plus difficiles de la programmation. Cela conduit souvent à une utilisation très restreinte du son, les capacités sonores de la machine n'étant pas totalement exploitées. C'est particulièrement le cas pour un ordinateur comme le CPC 464, qui possède un générateur de son performant et qui peut donc, en première approche, sembler déconcertant.

Le secret du succès d'un générateur de son réside dans l'exécution de quelques programmes simples, qui montrent clairement ce que chaque instruction fait réellement : un cetain nombre de ces programmes, ainsi que ce qu'ils réalisent, sont dans ce chapitre. Il est intéressant de les tester, c'est la seule manière de maîtriser parfaitement les possibilités sonores de votre appareil. Même si vous comprenez parfaitement chaque aspect des instructions associées à la partie sonore de l'appareil, il vous sera difficile (voire impossible) à moins d'avoir une grande expérience de la machine, d'associer un jeu de paramètres, aux sons qu'ils produisent. Même si vous possédez cette expérience, il sera souvent nécessaire de modifier votre programme original, de façon à obtenir le son désiré.

Le son du CPC 464 est reproduit par l'intermédiaire d'un haut-parleur qui se trouve dans l'ordinateur et non pas dans le moniteur. Il y a un réglage d'amplitude sur le côté de la machine.

Un niveau d'amplitude respectable est disponible à la

sortie du haut-parleur, ce niveau est suffisant pour des utilisations classiques. Il y a, à l'arrière de la machine, une prise (prise I/0) qui permet de se connecter à un ampli *Hi-fi* ou autre, pour obtenir un son plus fort. Cette connection se fait par l'intermédiaire d'une fiche stéréo 3,5 mm. Le signal est un signal stéréo à 3 voies. Le niveau de sortie est assez faible, et il se peut que certains amplis ne fonctionnent pas parfaitement. Bien que la prise de sortie s'adapte bien aux fiches existant sur les écouteurs des unités stéréo, il n'est malheureusement pas possible de les utiliser, le résultat n'étant pas de bonne qualité.

SOUND

Comme on pouvait l'attendre, la principale instruction relative au générateur de son est SOUND (*son* en Français). Il peut être utilisé avec d'autres instructions, mais de nombreux effets sonores sont possibles, simplement avec l'instruction SOUND, et il est préférable de maîtriser cette instruction avant d'essayer d'en utiliser d'autres.

SOUND est suivi de sept paramètres, mais tous ne sont pas toujours demandés. Le premier chiffre sélectionne la voie ou les voies auxquelles l'instruction est destinée. Cela se fait de la manière suivante :

1 sélectionne la voie A
2 sélectionne la voie B
4 sélectionne la voie C

De la sorte, si vous voulez que le son soit généré dans la voie B, le premier paramètre utilisé sera 2 ; si le son est créé sur les trois voies, on utilisera le nombre 7 (7=1+2+4). Ce paramètre a d'autres emplois, mais arrêtons-nous là pour l'instant. Le paramètre suivant règle la fréquence du son.

De même que sur la plupart des micro-ordinateurs, le fontionnement du générateur de son va à l'opposé du sens commun, puisque des valeurs élevées donnent des sons graves et vice versa. C'est une conséquence inévitable de la

manière d'opérer du générateur de son (à moins qu'un logiciel supplémentaire ne change les choses, ce qui est rarement le cas). Le générateur de son utilise l'horloge 4 MHz de l'ordinateur et divise cette fréquence par 16, soit 125 KHz. Ce nombre est alors divisé par le paramètre considéré. Des valeurs élevées de ce paramètre donnent donc des fréquences basses. Si ce paramètre vaut 4 ou moins, la fréquence est supérieure à la limite de ce que peut entendre l'homme ; en fait, il est nécessaire que ce paramètre ait une valeur comprise entre 5 et 8. De l'autre côté de la gamme, la valeur maximum est 4 095 qui correspond à une fréquence de 30 Hz, fréquence très basse. Le petit haut-parleur qui se trouve à l'intérieur de l'ordinateur ne reproduit pas parfaitement les fréquences très basses, il en résulte un bourdonnement.

Le troisième paramètre fixe la durée du son en centièmes de seconde. Il permet un contrôle très précis d'effets sonores comme certains airs musicaux. Le nombre maximum est 32 767, ce qui est largement suffisant en pratique puisqu'il correspond à une durée de plus de 5 minutes.

L'amplitude est donnée par le quatrième paramètre. Lorsque l'on utilise une instruction sonore (sans instruction ENV) ce chiffre varie de 0 où les voies sont coupées à 7 où l'amplitude est maximum. Nous nous intéresserons plus tard à la commande ENV, mais utilisée avec une instruction SOUND, la gamme d'amplitude s'étend de 0 à 15.

On peut ignorer pour l'instant les trois derniers paramètres, en leur donnant la valeur 0. L'instruction SOUND 1,250,100,6,0,0,0 donnera sur la voie A un son de fréquence 500 Hz (125 KHz divisé par 250) d'une durée de 1 seconde et d'amplitude 6. Il est possible d'utiliser 2 ou 3 voies pour améliorer légèrement le signal, mais la différence n'est pas très sensible si l'on utilise la même fréquence sur les 3 voies ; ce petit programme l'illustre :

```
10 SOUND 1,200,100,6,0,0,0
20 SOUND 7,200,100,6,0,0,0
```

On obtient simplement sur la voie A un son d'une seconde

suivi d'un son de même durée et de même fréquence sur les trois voies. La seule différence notable est l'augmentation du volume sonore.

Le listing suivant débute par une instruction qui incrémente la valeur du deuxième paramètre de 1 jusqu'à 4 095, valeur maximum possible. En d'autres termes, le son produit décrira toute la gamme de fréquence.

```
10 FOR P=1 TO 4095
20 SOUND 1,P,1,7,0,0,0
30 NEXT P
```

Si vous faites exécuter ce programme, vous remarquerez qu'au début, il y a une chute brutale de tonalité suivie d'une régression progressive. Cela provient du système employé, car une variation de 4 à 5 (facteur divisant 125 KHz) est beaucoup plus sensible qu'une variation de 4 000 à 4 001. Cependant, la gamme de sons aigus est très étendue.

Des effets sonores simples peuvent être obtenus en faisant varier la fréquence. Par exemple, voici un programme qui produit un son *Zap*.

```
10 FOR P=20 TO 80
20 SOUND 1,P,1,7,0,0,0
30 NEXT P
```

Le second paramètre varie de 20 à 80 par pas de 1 centième de seconde, de façon à ne pas entendre cette variation. Il est possible d'améliorer le son *Zap* en mettant une amplitude sonore élevée au départ, puis en la diminuant avec la fréquence. Cette variation d'amplitude porte le nom de formation d'enveloppe (en anglais : *envelope shaping*). Un tel son *Zap* est réalisé par le programme suivant :

```
10 V=7
20 FOR P=20 TO 80
30 SOUND 1,P,1,V,0,0,0
40 V=V-0.1
50 NEXT P
```

Ce programme ressemble beaucoup au précédent, cependant la ligne 10 initialise le volume tandis que la ligne 40 le décrémente de 0,1 à chaque réalisation de la boucle. On peut penser que ce programme ne fonctionne pas puisque V (paramètre d'amplitude) n'est pas entier. En réalité il n'y a pas d'erreur, car le générateur de son convertit V en entier, et il n'est donc pas nécessaire de rajouter une ligne de programme.

MUSIQUE

Il est assez facile de programmer de la musique, car l'ordinateur peut créer 3 harmonies différentes. Cependant, à moins d'être un grand connaisseur, il est peut être préférable de se limiter aux lignes mélodiques simples. C'est pourquoi, si vous voulez produire autre chose qu'un air simple, pour accompagner votre programme, il semble préférable d'utiliser des programmes spéciaux, plutôt que des programmes Basic. Le CPC 464 couvre une gamme de notes très étendue, sur 8 octaves. Cependant, comme nous l'avons déjà vu, le nombre de hautes fréquences est relativement limité, et les basses fréquences passent mal dans le haut-parleur. C'est pourquoi les 2 octaves supérieurs ainsi que l'octave inférieur ne sont pas très justes et donnent un son peu musical. Il est cependant possible de se restreindre aux 5 autres octaves disponibles pour programmer de la musique. L'ordinateur ne peut être programmé directement à l'aide de signes musicaux, la table qui se trouve en appendice du manuel du CPC 464 convertit les notes en valeur du paramètre de fréquence.

Le programme ci-dessous est utilisable pour programmer de courtes lignes mélodiques.

```
10 P=1
20 WHILE P()0
30 READ P,D,V
40 SOUND 1,P,D,V,0,0,0
50 WEND
60 DATA 239,50,5,213,50,5,190,50,5,179,50,5,159,50,
5,142,50,5,127,50,5,119,100,7,0,0,0
```

Une instruction WEND ... WHILE est utilisée pour boucler tant que la mélodie n'est pas terminée. La fréquence, la durée et l'amplitude sont lues dans cet ordre, dans l'instruction DATA en ligne 60. Les paramètres doivent donc être groupés par trois, et correspondent à l'ordre précisé ci-dessus. Le programme est terminé lorsque le deuxième paramètre vaut 0 ; cependant il est nécessaire d'attribuer aux paramètres de durée et amplitude une valeur (ici 0) sinon le programme chute en ligne 30 et on obtient un message d'erreur. Les diverses valeurs écrites en ligne 60 correspondent à un ton montant en C majeur, les nombres utilisés pour la durée et l'amplitude sont 50 et 5 pour toutes les notes sauf la dernière où ils valent 100 et 7. On peut jouer tous les tons possibles en leur attribuant les bonnes valeurs. Il est préférable, si l'on veut programmer beaucoup de notes, d'utiliser plusieurs instructions DATA pour faciliter la vérification et la correction. Il ne paraît pas nécessaire d'inclure dans le programme un réglage de l'amplitude, si ce n'est pour améliorer la mélodie ; cependant ce réglage permet des interruptions entre les notes. C'est essentiel : lorsque 2 ou 3 notes identiques se suivent, sans une courte pause entre chacune, on entend une seule note de longue durée, ce qui n'est pas l'effet attendu.

RENDEZ-VOUS

Jusqu'à présent nous n'avons considéré le premier paramètre de l'instruction SOUND, que comme un sélectionneur de voies, mais il a d'autres fonctions ; une d'entre elles est la possibilité de rendez-vous de plusieurs voies entre elles. En d'autres termes, une ou deux voies attendent une instruction SOUND pour que la troisième voie soit reçue ; dès lors les instructions SOUND — qui permettent ces rendez-vous — démarrent simultanément. Voici les nombres correspondant aux rendez-vous.

8 Rendez-vous avec la voie A
16 Rendez-vous avec la voie B
32 Rendez-vous avec la voie C

Ce programme va vous aider à mieux comprendre la manœuvre.

```
10 SOUND 12,100,50,5,0,0,0
20 FOR D=1 TO 1000 : NEXT D
30 SOUND 33,200,50,5,0,0,0
```

La ligne 10 comporte une instruction SOUND, qui réalise un rendez-vous entre les voies C et A. Le premier paramètre (le paramètre de choix des voies) a la valeur 12, qui correspond à 4 pour sélectionner la voie C, plus 8 pour lui donner rendez-vous avec la voie A. Cette instruction n'est pas exécutable tant qu'une instruction SOUND relative à la voie A n'est pas rencontrée. La ligne 20 crée simplement un délai, alors que la ligne 30 est une instruction SOUND concernant la voie A. Notons que la valeur du paramètre est 33, et non pas 1, puisque 1 sélectionne la voie A et 32 permet le rendez-vous avec la voie C. Le rendez-vous est possible seulement si toutes les voies concernées ont les bons paramètres. Si vous exécutez ce programme, il doit y avoir une pause (d'environ 1 s) suivie simultanément des instructions SOUND.

Le programme suivant montre une utilisation pratique du rendez-vous.

```
10 SOUND 1,120,50,5,0,0,0
20 SOUND 2,50,100,5,0,0,0
30 SOUND 17,100,50,5,0,0,0
40 SOUND 10,150,50,5,0,0,0
```

Les deux premières lignes créent des sons sur les voies A et B, de durées différentes puisque celui de la voie A dure une demi-seconde et celui de la voie B une seconde. Les deux instructions suivantes concernent encore les voies A et B, mais en vue de créer des rendez-vous. Cela signifie que la voie A ne donnera sa seconde note que lorsque la voie B en fera de même. Cela s'entend très bien lors de l'exécution du programme.

Ceci est important puisque lorsque vous écrivez 2 ou 3

harmonies, il n'est pas nécessaire qu'elles aient le même rythme : il est possible en effet de programmer des temps de repos. Le système de rendez-vous permet aussi de s'assurer du synchronisme des lignes musicales.

HOLD/RELEASE

Un autre emploi du paramètre de choix des voies est HOLD/RELEASE. *Hold* (*saisir* en français) est obtenu en ajoutant 64 au paramètre. Le programme suivant montre le déroulement de cette opération

```
10 SOUND 65,100,100,5,0,0,0
20 SOUND 66,150,100,5,0,0,0
30 SOUND 68,250,100,5,0,0,0
40 FOR D=1 TO 1000:NEXTD
50 RELEASE 7
```

Les 3 instructions SOUND des lignes 10 à 30 correspondent respectivement aux trois voies A, B, C. Mais, puisque l'on a additionné 64 au paramètre pour *saisir* ces voies, les instructions ne seront pas exécutées lorsqu'elles seront rencontrées par l'ordinateur. La ligne 40 crée un délai d'une seconde avant que l'instruction RELEASE (*libération* en français) ne soit rencontrée en ligne 50. L'instruction RELEASE permet de libérer les diverses voies en permettant l'exécution des instructions SOUND. On utilise les nombres suivants, pour sélectionner les différentes voies dans l'instruction RELEASE.

1 sélectionne la voie A
2 sélectionne la voie B
4 sélectionne la voie C

Si l'on doit libérer plusieurs voies, on utilise la somme des chiffres appropriés. Dans notre exemple le chiffre 7 libère les trois voies.

DEBORDEMENT ***(FLUSHING)***

Le dernier emploi du premier paramètre est le débordement *(flushing)*, il s'obtient en ajoutant 128 à notre paramètre. Ce court programme illustre cette opération.

```
10 FOR C = 1 TO 60
20 SOUND 129,1,1,0,0,0,0
30 NEXT C
```

Les lignes 10 et 30 constituent une boucle. Celle-ci est réalisée 60 fois, le paramètre de durée de l'instruction SOUND permet d'obtenir un son d'une seconde à chaque réalisation de la boucle. Cependant le paramètre de choix des voies de valeur 129 crée sur la voie A un débordement (1 pour la voie A plus 128 pour le débordement). L'effet de débordement est la réalisation immédiate de l'instruction quelles que soient les instructions précédentes. Si vous exécutez ce programme, vous remarquerez que l'instruction ne donne sa valeur réelle qu'après la dernière boucle, les autres durant le temps du bouclage. Comme le Basic utilisé dans le CPC 464 est rapide, la durée totale de ce petit programme ne dépasse guère la seconde alors qu'elle serait d'une minute sans débordement.

On utilise le débordement pour couper une voie du générateur de son, comme dans l'exemple suivant

```
10 SOUND 65,200,100,7,0,0,0
20 SOUND 129,1,1,0,0,0,0
30 RELEASE 1
```

La ligne 10 crée un son d'une seconde mais que l'on a saisi. La ligne 20 coupe la voie A par débordement puisqu'aucun son n'est produit, le paramètre de volume valant 0. Finalement la ligne 30 libère la voie A, mais aucun son ne doit être émis puisque le débordement en ligne 20 court-circuite les instructions de la ligne 10.

BRUIT *(NOISE)*

Un signal de bruit peut-être généré et mélangé aux sons des différentes voies, cependant il y a un seul générateur de bruit et on ne peut produire un seul bruit à la fois. Il n'est pas nécessaire de produire un son, pour accompagner le bruit. L'emploi de hautes fréquences inaudibles permettent d'isoler le bruit en sortie. Le type de bruit créé est le classique sifflement, dont la fréquence peut varier. On utilise le bruit avec un paramètre de valeur compris entre 1 et 31, c'est le dernier paramètre de l'instruction SOUND (une valeur nulle signifie une absence de bruit). 31 correspond au ton minimum et 1 au maximum. Une gamme de 31 valeurs peut sembler restrictive, mais l'utilisation principale du bruit résidant dans les effets sonores, cette gamme disponible est en fait très suffisante.

Le bruit est très employé pour générer des sons d'explosion. Le programme ci-dessous crée un tel son

```
10 P=12
20 FOR V=7 TO 0 STEP-1
30 SOUND 1,1,20,V,0,0,P
40 P = P + 2
50 NEXT V
```

Un son est créé en ligne 30 sur la voie A, mais le ton est trop élevé pour qu'il soit audible. Sa durée est d'un cinquième de seconde à chaque réalisation de la boucle. L'amplitude et le bruit sont contrôlés pour les variables respectives V et P. V varie de 7 à 0 pendant le déroulement du programme, 7 étant l'amplitude maximale. Cette variable est nécessaire à la bonne reproduction du bruit d'une explosion. La ligne 40 incrémente le paramètre de bruit de 2 à chaque boucle, ceci provoque une diminution des aigus et une atténuation très importante en fin du signal.

Ce programme est facilement modifiable pour simuler le bruit du coup de revolver ; pour cela il faut utiliser un son plus court et plus aigu. Voici la version modifiée de ce programme.

```
10 P=3
20 FOR V=7 TO 0 STEP-1
30 SOUND 1,1,3,V,0,0,P
40 P=P +1
50 NEXT V
```

ENV

Dans bien des cas, une instruction SOUND, avec éventuellement une boucle ou tout autre support est suffisante pour créer l'effet désiré. Cependant lorsque des formes d'amplitude ou de fréquence complexes sont nécessaires, il est préférable d'utiliser une instruction SOUND avec une instruction ENV (enveloppe-volume) ou ENT (enveloppe-ton). Considérons tout d'abord ENV.

Le premier paramètre employé dans une instruction ENV est simplement un nombre d'identification (de 1 à 15) qui permet d'appeler l'ENVeloppe désirée, dans une instruction SOUND. Ce nombre est le cinquième paramètre de l'instruction SOUND. Les trois autres nombres de l'instruction ENV contrôlent le nombre de pas, l'augmentation ou la diminution d'amplitude entre chacun de ces pas, et la durée de chaque pas (en centièmes de seconde). Le nombre de pas doit être compris entre 0 et 127 alors que l'écart d'amplitude entre chaque pas varie de −128 à +127. Cependant, il faut noter qu'il y a seulement 15 niveaux d'amplitude (plus le niveau 0) ; c'est pourquoi l'écart d'amplitude doit normalement être compris entre −15 et +15. La durée maximale du pas est 255. Par exemple, pour diminuer l'amplitude de 15 à 0 en 15 pas d'une durée d'un dixième de seconde, il faudra écrire

```
ENV 1,15,-1,10
```

L'enveloppe utilisée s'indentifie à l'enveloppe 1.

En pratique on peut utiliser jusqu'à 5 sections pour la forme d'une enveloppe, chaque section étant déterminée par trois chiffres : le nombre de pas, la taille du pas et sa durée.

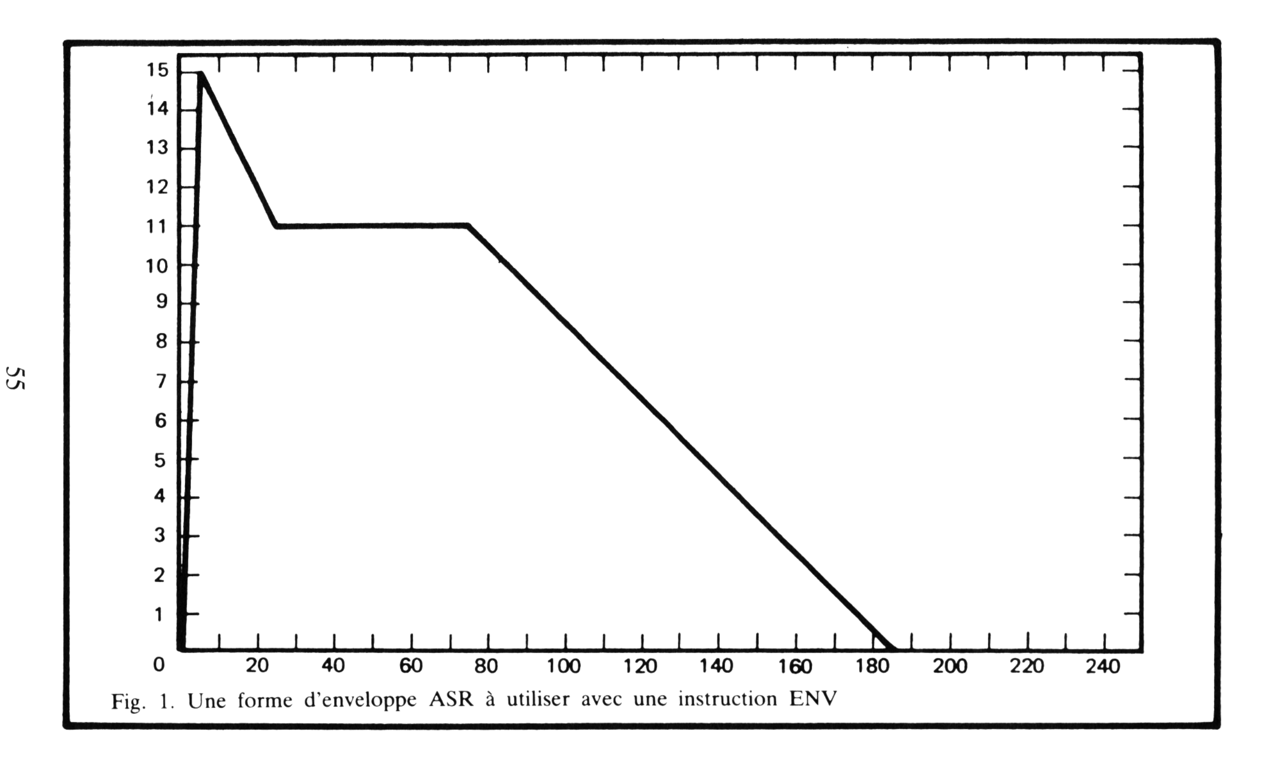


Fig. 1. Une forme d'enveloppe ASR à utiliser avec une instruction ENV

On obtient des enveloppes complexes, et parfois un peu difficiles à calculer, c'est pourquoi il est préférable de dessiner l'enveloppe avant de calculer rigoureusement les nombres de l'instruction ENV. La figure 1 illustre ce propos.

Cette enveloppe essaye de reproduire celle d'un piano ainsi que d'autres instruments. Pour cela, il faut une très rapide progression de l'amplitude suivie d'une diminution rapide jusqu'à un niveau moyen pendant lequel le son est constant, puis diminue progressivement. Cette forme d'enveloppe porte le nom de ADSR (Attack : "attaque", Decay : "déclin", Sustain : "soutien", Release : "libération"). En figure 1, le son atteind son niveau maximum en 5 centièmes de seconde. Ce niveau étant de 15, il faut utiliser 5 pas de durée minimum, la taille du pas devant être + 3. La première partie de l'instruction ENV s'écrit donc :

```
ENV, 1,5,3,1
```

La seconde section dure 20 centièmes de seconde, l'amplitude chute de quatre unités. Il faut donc 4 pas de durée 5 centièmes de seconde, l'amplitude diminuant de 1 par pas.

La section suivante n'utilise qu'un seul pas, sans variation d'amplitude de durée 50 centièmes de seconde. Enfin la dernière section demande 11 pas de 10 dixièmes de seconde, l'amplitude diminuant de 1 par pas. Si vous construisez une instruction ENV qui décrit la forme d'enveloppe ci-dessus, elle aura la structure suivante.

```
ENV 1,5,3,1,4,-1,5,1,0,50,11,-1,10
```

Elle peut être utilisée dans un programme musical comme dans l'exemple ci-dessous :

```
10 ENV 1,5,3,1,4-1,5,1,0,50,11,-1,10
20 READ P,D
30 IF P=0 THEN 70
40 SOUND 1,P,D,0,1,0,0
```

```
50 GOTO 20
60 DATA 239,50,213,50,190,50,179,50,159,50,142,50,127,
   50,119,185,0,0
```

Ce programme est similaire au programme musical précédent, et les valeurs de l'instruction DATA en ligne 60 correspondent à la même ligne musicale. Les valeurs de l'instruction DATA sont groupées par paires : les notes et leurs durées. Il n'est pas nécessaire d'introduire le paramètre d'amplitude dans l'instruction DATA puisque l'enveloppe assure la séparation des notes de même valeur. En fait, dans cet exemple, l'amplitude ne peut être modifiée à partir de l'instruction DATA puisque le paramètre correspondant vaut 0. Sinon le niveau de départ et d'arrivée à 0 n'étant pas spécifié par l'instruction ENV, le paramètre d'amplitude dépasserait la limite possible.

Il est important de remarquer que la durée de chaque note est fixée par l'instruction SOUND. Ceci n'est possible que si cette durée est inférieure à celle fixée par l'instruction ENV, ou bien si l'instruction ENV donne une valeur finale d'amplitude non nulle. Sinon le son se termine quand l'amplitude atteint la valeur zéro en fin de l'instruction ENV.

ENT

ENT opère de la même manière que ENV, et comme précédemment, le premier paramètre compris entre 1 et 15, sert à identifier l'instruction ENT. Elle est appelée par une instruction SOUND à l'aide du sixième paramètre de cette instruction. De la même manière trois nombres définissent les caractéristiques de chaque section de l'enveloppe et cinq sections peuvent être utilisées.

- Le premier nombre fixe le nombre de pas (de 0 à 239),
- le deuxième, la variation d'unités de tons entre chaque pas (de − 128 à + 127)
- le troisième, la durée de chaque pas (de 0 à 255).

Prenons un exemple simple ; il imite le son d'une sirène de voiture de police.

```
10 ENT -1,20,2,1,20,-2,1
20 SOUND 1,50,1000,6,0,1,0
```

Notons tout d'abord que le chiffre qui identifie l'enveloppe étant négatif, l'instruction est alors répétée automatiquement. L'emploi d'un nombre positif ne permet qu'une seule exécution de l'enveloppe, et la fréquence reste à sa valeur finale, une fois l'instruction SOUND terminée. On utilise un nombre d'enveloppes positif dans l'instruction SOUND.

Les valeurs de l'instruction ENT permettent de faire varier le ton de +2 à chaque pas ; il y a 20 pas de un centième de seconde. Le deuxième jeu de 3 paramètres inverse le processus et rétablit le ton à sa valeur initiale. Cette valeur est fixée par l'instruction SOUND. Dans notre exemple, elle augmente de 40, puis diminue de 40, et ainsi de suite.

Si le nombre déterminant la durée, fixé par l'instruction SOUND, est trop court pour permettre la réalisation complète de l'enveloppe, le signal s'achève néanmoins après le temps spécifié par l'instruction SOUND. Il est parfaitement possible d'utiliser les instructions ENV et ENT ensemble, avec une seule instruction SOUND. Par exemple, le programme musical précédent peut être modifié en ajoutant la ligne 5 et en changeant la ligne 40.

```
 5 ENT - 1,1,1,4,2,-1,4,1,1,5
40 SOUND 1,P,D,0,1,1,0
```

L'instruction ENT a pour effet de faire varier le ton de plus ou moins 1, en vue de créer un effet de vibrato.

Chapitre 6

GRAPHISME 1 — MODES ET COULEURS

La plupart des ordinateurs qui utilisent différents modes d'écran, distinguent nettement le nombre de couleurs disponibles du mode, et la quantité de détails visibles ; beaucoup de détails signifie moins de couleurs.

L'Amstrad possède trois modes numérotés de 0 à 2. Le mode 0 a la plus faible résolution, mais le plus grand nombre de couleurs ; il peut inscrire 20 colonnes de texte (tous les modes ont 25 lignes de texte) et utilise jusqu'à 16 couleurs. La résolution graphique est de 160 × 200, ce qui signifie qu'il peut inscrire 160 points séparés de gauche à droite de l'écran, et 200 de bas en haut. Le nombre important de couleurs disponibles simultanément en font un mode privilégié pour les jeux, où les détails précis ont généralement peu d'importance.

Le mode 1 est le mode utilisé par défaut, celui qui apparaît lorsqu'on branche l'appareil. Il dispose de 40 colonnes de texte et a une résolution graphique de 320 × 200 ; il peut utiliser 4 couleurs en même temps. C'est le mode utilisé pour les programmes généraux et les graphiques scientifiques ou industriels, dans lesquels les détails importent plus que les couleurs. Quatre couleurs sont alors nécessaires.

Le mode 2 possède 80 colonnes de texte et a une résolution graphique de 640 × 200. Il n'a que deux couleurs disponibles à la fois. Ce mode est utilisé dans le graphisme de haute précision, et pour les applications comme le traitement de texte, ou l'agrandissement. Cependant, si les deux couleurs ne sont pas choisies judicieusement, il est possible que le texte soit peu visible sur le moniteur couleur, et même pas visible du tout sur l'écran T.V. utilisé avec un modulateur spécial.

L'Amstrad peut produitre 27 couleurs. Vous pouvez choisir parmi ces couleurs, le nombre de choix étant imposé par le mode utilisé. Il serait indentique d'avoir 27 bouteilles d'encre pour remplir vos stylos. Vous avez 2 stylos pour

écrire très petit, 4 pour écrire normalement, et 16 pour écrire en gros caractères. Vous pouvez remplir vos stylos avec l'encre que vous désirez (il est possible d'utiliser 2 stylos ou plus de la même couleur). La couleur du papier utilisé doit être la même que l'encre de l'un des stylos. Bien entendu, si vous désirez écrire sur du papier, vous pouvez remplir votre stylo avec des bouteilles d'encre différentes. Cette opération est possible sur Amstrad : si vous l'utilisez, toutes les inscriptions déjà faites changeront alors de couleurs. Cela peut être très utile en programmation.

Cette analogie est intéressante, puisque les mots-clés utilisés pour la couleur, dans Locomotive Basic, sont INK *(encre)* PEN *(stylo)* et PAPER *(papier)*. INK attribue l'une des 27 couleurs à l'un des stylos disponibles, PEN sélectionne le stylo utilisé pour écrire (jusqu'à la rencontre d'une nouvelle commande PEN) et PAPER indique le stylo utilisé pour l'écran.

En fait tous les stylos ont des valeurs, et il n'est pas forcément nécessaire d'utiliser les instructions INK. Vous pouvez utiliser les couleurs attribuées par l'ordinateur. Prenez votre ordinateur et inscrivez *"pen 2" :* lorsque vous appuyez sur ENTER le message "READY" apparaît en bleu. Ecrivez *"pen 3"* (en bleu aussi), le message "READY" sera rouge. Le mode 1 attribue aux stylos les couleurs suivantes : bleu foncé *(pen 0),* jaune *(pen 1),* bleu clair *(pen 2),* rouge *(pen 3). Pen 0* est utilisé pour le fond de l'écran (PAPER), et *Pen 1* pour les inscriptions, tant qu'aucune commande n'est intervenue.

Si vous tapez *"paper 1"*, le texte apparaîtra en rouge sur fond jaune. Si vous colorez maintenant l'écran en rouge (c'est-à-dire *paper 3*), le texte imprimé sera invisible.

Pour illustrer le changement d'encre dans un stylo, réinitialiser l'ordinateur (SHIFT/CTRL/ESCAPE), puis écrivez *"ink 0,24"*. L'écran deviendra jaune ; bien sûr, des caractères jaunes seront invisibles ! Ecrivez alors "ink 1,1", vous ne verrez pas ce que vous inscrirez, mais vous devrez y arriver. Lorsque vous appuyez sur ENTER, les caractères apparaissent en bleu foncé ; ils peuvent cependant sembler noirs. INK 1,2 donnera du bleu clair. Le premier chiffre de la

commande INK est le numéro du stylo à remplir, le deuxième chiffre la couleur de l'encre (0 à 26).

En fait, vous pouvez attribuer une couleur aux 16 stylos, peu importe le mode utilisé, mais vous n'avez qu'un nombre limité de stylos. C'est-à-dire *pen 0* et *pen 1* en mode 2, *pen 0* à *3* en mode 1 et *pen 0* à *16* (soit tous les stylos) en mode 0. Ainsi le stylo n'est pas changé par le mode ; si vous prenez *pen 3* en mode 1, puis en mode 0, le texte sera toujours écrit en rouge.

Si vous utilisez un numéro de stylo supérieur au nombre de stylos du mode, il sera transformé en nombre modulo de stylos du mode. Ainsi, en prenant *pen 4* en mode 1, vous obtenez *pen 0*, et *pen 5* donnera *pen 1*. Ceci n'est pas changé par le mode : si vous utilisez *pen 6* en mode 1, vous obtenez *pen 2* ; si vous passez en mode 0, *pen 2* reste valable. Par contre, si vous voulez le mode 2, il faut une autre réduction et l'on obtiendra *pen 0*. (Dans cet exemple, comme c'est le cas lorsqu'on ne définit pas le mode initial, le texte est invisible, puisque de même couleur que PAPER.) Le listing 11 montre un exemple des possibilités des couleurs du mode 0. La première boucle des lignes 50 à 80 inscrit sur l'écran une ligne de texte dans chacune des 16 couleurs. La ligne correspondant à 0 ne peut être lue, puisqu'elle est de la même couleur que l'écran. La seconde partie du programme des lignes 90 à 140 est constituée de deux boucles FOR... NEXT qui permettent aux 16 stylos de prendre les 26 couleurs. En fin de boucle tous les stylos sont de couleur 26. La fin du programme redonne aux quatre stylos de mode 1 leur valeur par défaut, de façon à voir ce que l'on fait. Vous pouvez essayer de modifier le programme pour réinitialiser les 16 stylos (indication : utiliser une boucle FOR... NEXT et des instructions DATA, ou bien vous pouvez simplement remplacer les lignes 140 à 210 par CALL & BBFF : c'est un sous-programme qui réinitalise les couleurs d'écran et le mode, mais pas la sélection des stylos utilisés).

La commande INK permet aussi de produire sur n'importe quel stylo un effet de *flash*. C'est possible en donnant deux couleurs d'encre qui apparaissent alternativement. Il est

possible de faire apparaître et disparaître des caractères en prenant, pour l'une des couleurs, celle de l'écran.

L'effet de *flash* est contrôlé par la commande SPEED INK, suivi de deux nombres entiers. Le premier donne le temps d'apparition de la première couleur, le second celui de la deuxième, on peut ainsi contrôler à la fois la vitesse du *flash* et la durée relative des couleurs. Ces commandes sont faciles d'accès et leur utilisation est recommandée, mais ne soyez pas trop tape-à-l'œil !

LISTING 11

```
2Ø REM changement de couleur
3Ø MODE Ø
4Ø CLS
5Ø FOR colour=Ø TO 15
6Ø PEN colour
7Ø PRINT "Voici un stylo";colour
8Ø NEXT colour
9Ø FOR quill=Ø TO 15
1ØØ FOR colour=Ø TO 26
11Ø INK quill,colour
12Ø FOR delay=1 TO 2ØØ:NEXT delay
13Ø NEXT colour
14Ø NEXT quill
15Ø MODE 1
16Ø INK Ø,1
17Ø INK 1,24
18Ø INK 2,2
19Ø INK 3,3
2ØØ PEN 1
21Ø END
```

TRACÉ DE LIGNES

Amstrad a un important jeu de commandes pour tracer les lignes et inscrire des points sur l'écran. Il utilise un système

de coordonnées qui donne la position horizontale et verticale.

Bien que la résolution soit fonction du mode, les valeurs des coordonnées restent les mêmes. Cela signifie que le jeu de commandes DRAW (*dessin* en français) dessinera sur l'écran la même forme à la même place quel que soit le mode. C'est un avantage considérable. Bien sûr le nombre de détails visibles et la finesse des traits sont différents.

En fait, les trois modes ont la même résolution verticale, soit 200 lignes. Le système de coordonnée utilise une échelle verticale de 400, cela signifie que les points de coordonnée 0 et 1 auront la même place sur l'écran, de même que 2 et 3, 3 et 4, et ainsi de suite. C'est pourquoi il n'est pas nécessaire d'utiliser des nombres impairs pour les coordonnées verticales.

La coordonnée horizontale utilise 640 points. Ceci dans le mode de plus grande résolution, c'est-à-dire le mode 2. Le mode 1 utilise 320 points horizontalement donc les points 0 à 1 donneront le même point sur l'écran, comme en coordonnée verticale. Le mode 1 a la même résolution dans les deux directions, cela peut être utile dans certains graphiques. Le mode 2 a 160 points horizontalement, les points 0, 1, 2, 3 correspondent donc au même endroit de l'écran.

Les coordonnées graphiques sur l'écran sont comprises entre 0 et 639 horizontalement et 0 et 399 verticalement. La position 0, 0 appelée origine se trouve en bas à gauche de l'écran (contrairement à l'origine du texte qui est en haut à gauche), mais elle peut être déplacée partout sur l'écran (ou même en dehors) grâce à la commande ORIGIN *(origine)*.

Avec Amstrad, il est possible de dessiner en dehors de l'écran. Les positions extérieures à l'écran sont enregistrées, ce qui signifie que si vous tracez une ligne à l'extérieur, puisque vous retournez sur l'écran, l'angle de la seconde ligne reste correct. Cela facilite parfois la création d'effets de perspective.

L'ordinateur a un curseur graphique, semblable au curseur de texte, mais il est invisible. Lorsque les lignes sont tracées, elles le sont à partir de la position initiale du curseur jusqu'à sa position finale.

Il y a deux manières de spécifier les coordonnées, la manière relative et la manière absolue. Relatif signifie que les valeurs correspondent à un déplacement à partir de la position actuelle du curseur. La forme absolue donne les déplacement à partir de l'origine, l'écran étant considéré comme une grille imaginaire. Ces deux formes permettent des coordonnées négatives.

Dans l'ensemble, il est probable que vous trouviez le système absolu plus simple et plus utile, mais les coordonnées relatives simplifient parfois les programmes.

Les trois commandes graphiques principales sont MOVE, DRAW et PLOT. Ce sont les formes absolues. Les formes relatives prennent un R en fin de mot : MOVER, DRAWR, PLOTR. MOVE déplace un curseur graphique sans inscription sur l'écran. DRAW trace une ligne de la position initiale du curseur jusqu'à la nouvelle position spécifiée. PLOT est semblable à MOVE mais il inscrit un point à l'endroit de la nouvelle position du curseur.

Le Listing 12 utilise MOVE et DRAW ainsi que des coordonnées relatives et absolues. Il dessine un rectangle sur l'écran, de taille et de positions déterminées par les instructions INPUT des lignes 50 à 70. La partie du programme qui concerne la création du rectangle est écrite sous forme de sous-programme que vous pouvez utiliser dans vos propres programmes.

La ligne 1010 déplace le curseur jusqu'au point de coordonnées absolues définies en ligne 50. Les quatre lignes suivantes tracent le rectangle à l'aide de coordonnées relatives. L'ordre du tracé est le suivant : côté gauche, haut ; côté droit, bas. Notons que les déplacements de bas en haut et de gauche à droite sont positifs, ceux de haut en bas et de droite à gauche sont négatifs.

LISTING 12

```
2Ø REM tracé d'un rectangle
3Ø MODE 1
4Ø WHILE 1
```

```
5Ø INPUT "Position du coin inférieur gauche (x,y)";x,y
6Ø INPUT "hauteur";high
7Ø INPUT "largeur";wide
8Ø CLS
9Ø GOSUB 1ØØØ
1ØØ WEND
1ØØØ REM sous-programme de dessin du rectangle
1Ø1Ø MOVE x,y
1Ø2Ø DRAWR Ø,high
1Ø3Ø DRAWR wide, Ø
1Ø4Ø DRAWR Ø,-high
1Ø5Ø DRAWR -wide,Ø
1Ø6Ø RETURN
```

Locomotive Basic ne possède pas de commande pour tracer les cercles. Il est cependant assez facile de dessiner un cercle en utilisant les fonctions SIN et COS. Ceci est utilisé dans le Listing 13, qui trace un certain nombre d'ellipses à l'intérieur d'un cercle, en utilisant les couleurs du mode 1.

La ligne 30 donne la couleur de la frontière de l'écran. Cette frontière peut prendre les 27 couleurs possibles ou alterner entre deux, en utilisant un *flash*, ceci quels que soient les stylos utilisés et le nombre de couleurs du mode. La — ou les — couleurs de l'instruction BORDER sont choisies parmi la liste des 27 possibles et non parmi les stylos. Dans notre exemple, nous avons néanmoins choisi un stylo.

Les lignes 50 à 80 tracent un rectangle le long de l'écran. Remarquons l'emploi d'un paramètre supplémentaire en fin des instructions DRAW. La couleur utilisée pour le tracé des lignes n'est pas la même que celle du stylo qui sert à écrire. Elle est définie par ce paramètre supplémentaire dont la valeur doit être celle d'un stylo utilisable. Si la couleur n'est pas spécifiée par l'instruction DRAW, on utilise la dernière utilisée, et non pas la couleur de *pen 1* comme dans notre exemple.

Les lignes 100 à 140 tracent le cercle. En ligne 100, on définit x et y, les coordonnées du centre, et d le diamètre (en fait, ce nombre est le rayon et il aurait été préférable de

l'appeler r, mais d est utilisé depuis si longtemps qu'il est difficile de changer !). La ligne 110 déplace le curseur graphique vers le premier point du cercle, une boucle FOR... NEXT des lignes 120 à 140 permet le dessin. Notons que la ligne 130 sélectionne la couleur de *pen 2*.

Les lignes 160 à 250 tracent les diverses ellipses. Pour cela il faut modifier les cercles en les étirant dans une direction que l'on contrôle par la ligne 170 (le nombre qui divise PI donne le nombre d'ellipses) et par un autre paramètre que l'on définit en ligne 190 (100 dans l'exemple). Les lignes 200 à 230 effectuent le tracé. L'ordre de SIN et COS des lignes 200/210 est différent de celui de la ligne 130. Les ellipses sont donc tracées dans le sens des aiguilles d'une montre, contrairement au cercle. La ligne 220 déplace le curseur vers le point de départ de chaque ellipse. La ligne 230 sélectionne *pen 3*.

LISTING 13

```
2Ø MODE 1
3Ø BORDER 2
4Ø REM tracé du rectangle
5Ø DRAW Ø,399,1
6Ø DRAW 639,399,1
7Ø DRAW 639,Ø,1
8Ø DRAW Ø,Ø,1
9Ø REM tracé du cercle
1ØØ x=32Ø:y=2ØØ:d=196
11Ø MOVE x+d,y
12Ø FOR c=Ø TO 2*PI STEP Ø.Ø3
13Ø DRAW x+d*cos(c),y+d*SIN(c),2
14Ø NEXT c
15Ø REM tracé des ellipses
16Ø d=92
17Ø FOR e=Ø TO PI STEP PI/5
18Ø FOR p=Ø TO 2*PI STEP Ø.Ø3
19Ø f=1ØØ*COS(p)
2ØØ xl=x+f*SIN(e)+d*SIN(p+e)
```

```
210 yl=y+f*COS(e)+d*COS(p+e)
220 IF p=0 THEN MOVE xl,yl
230 DRAW xl,yl,3
240 NEXT p
250 NEXT e
260 END
```

Le listing 14 utilise PLOT. Ce programme inscrit simplement des points sur l'écran, à des positions et avec des couleurs différentes. Les résultats peuvent être jolis.

LISTING 14

```
20 MODE 0
30 WHILE 1
40 PLOT RND(1)*640,RND(1)*400,RND(l)*15
50 WEND
```

Pour conclure ce paragraphe, le listing 15 vous permettra de créer l'image représentée en couverture. C'est encore une variante du tracé du cercle. Dans cet exemple, la valeur de d est décrémentée (ligne 140) pour obtenir une spirale. La spirale n'est pas dessinée, le curseur se contentant de la parcourir (ligne 120). Une seconde spirale, liée à la première, est créée par la ligne 130 avec l'aide de la ligne 100. Les lignes sont tracées à partir des points de la première spirale pour aboutir sur les points de la seconde. La variable x contrôle la couleur. Les couleurs sont utilisées séquentiellement sauf celle de l'écran (ligne 80) qui est supprimée, de même que deux couleurs indésirables (ligne 90).

LISTING 15

```
20 MODE 0
30 BORDER 2
40 PAPER 5:CLS
```

```
5Ø x=1:d=35Ø
6Ø WHILE d>34
7Ø FOR c=Ø TO 2*PI STEP Ø.1
8Ø IF x=5 THEN x=6
9Ø IF x=14 THEN x=1
1ØØ e=d-5Ø
11Ø IF e<Ø THEN e=Ø
12Ø MOVE 32Ø+d*COS(c),2ØØ+d*SIN(c)
13Ø DRAW 32Ø+e*COS(c-Ø.5),2ØØ+e*SIN(c-Ø.5),x
14Ø d=d-1
15Ø IF e<Ø THEN e=Ø
16Ø x=x+1
17Ø NEXT c
18Ø WEND
19Ø GOTO 19Ø
```

WINDOWS *(FENÊTRES)*

Normalement, la totalité de l'écran du moniteur ou de la télévision, à partir de la frontière est disponible lors de l'opération PRINT ou LIST. Cependant, il est possible de diviser l'écran en sections et d'opérer PRINT ou LIST pour chaque section ou fenêtre *(window)*. Le CPC 464 peut créer jusqu'à huit fenêtres. Bien entendu, l'instruction WINDOW est utilisée pour créer une fenêtre, elle comporte cinq paramètres. Le premier d'entre eux sert à identifier chaque fenêtre, de manière à sélectionner la bonne lors d'une instruction PRINT. Les quatre autres donnent les dimensions de la fenêtre dans le système des coordonnées usuelles. Les deux premiers déterminent la largeur et la position latérale de la fenêtre puisqu'ils indiquent la colonne la plus à gauche et la plus à droite. De même les deux derniers nombres définissent la ligne la plus basse et la ligne la plus haute.

Par exemple

```
WINDOW ≠ 1,10,20,15,22
```

crée une fenêtre identifiée par le numéro 1 dont les coordonnées du coin en haut à gauche sont 10 et 15 et celles

du coin en bas à droite 20-22. Bien entendu, la surface exacte d'écran occupé dépend du mode utilisé puisque la numérotation des coordonnées en dépend.

DÉS

Le programme "dés" exposé dans le Listing 16 montre une manière d'utiliser les fenêtres. L'idée du programme est de dessiner une paire de dés sur l'écran. En appuyant sur la touche d'intervalle, on fait rouler les dés un instant, les numéros gagnants sont alors inscrits. Deux fenêtres sont utilisées pour créer la forme des dés.

La ligne 20 donne le mode, dans cet exemple, une grande résolution n'est pas nécessaire, mais il est intéressant d'avoir une grande gamme de couleurs, c'est pourquoi le mode 0 est choisi. La ligne 30 colorie l'écran en bleu clair alors que la ligne 40 dessine une lisière verte. Les fenêtres sont définies dans les deux lignes suivantes, elles portent les numéros 1 et 2. La figure 2 montre la taille et la position des fenêtres, de même que la position des points sur chaque dé. Chaque fenêtre a une largeur de 7 coordonnées et une hauteur de 12, mais comme chaque case élémentaire (en mode 0) est à peu près deux fois plus large que haute, les dés sont carrés. De même, chaque point du dé est un carré.

Les lignes 70 à 140 indiquent la couleur de l'écran et celle des stylos pour chaque fenêtre, grâce au paramètre d'identification. Le dé de droite est jaune à points oranges, alors que celui de gauche est magenta clair à points rouges. De toute façon, vous n'aurez pas de difficulté pour changer ces couleurs si vous ne les aimez pas. Les lignes 150 à 160 servent à effacer le contenu des fenêtres. Les deux lignes suivantes créent deux entiers aléatoires de 1 à 6 dont les valeurs sont rangées dans l et r (dé gauche et dé droit). Un bruit, imitant le roulement des dés est créé par la ligne 200.

Les points sont placés sur les dés grâce à une série d'instructions IF ... THEN, et un ensemble de sous-programmes. Quatre sous-programmes sont utilisés par

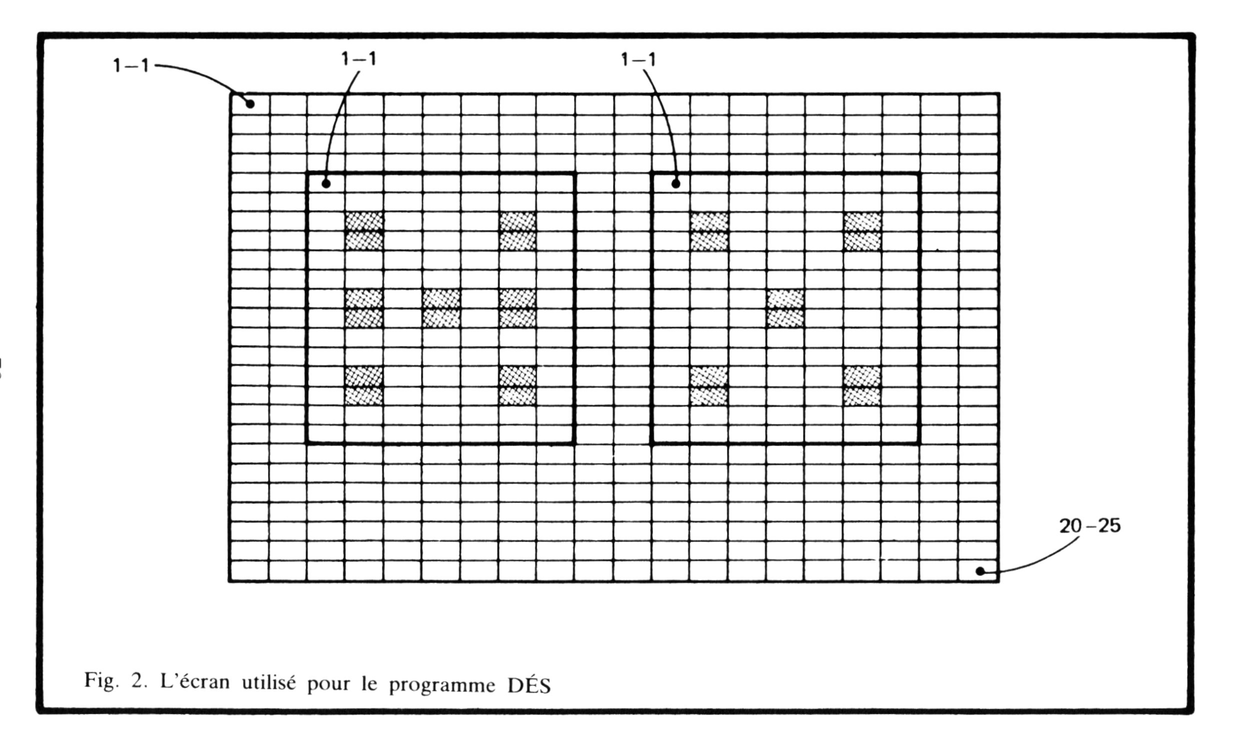


Fig. 2. L'écran utilisé pour le programme DÉS

chaque dé, le premier d'entre eux dessine le point du centre. Pour le dé gauche, ce sous-programme débute en ligne 400. Une nouvelle instruction est utilisée : LOCATE. Elle sert à positionner le curseur à l'intérieur d'une fenêtre. Elle est utilisée ici pour placer le curseur sur l'endroit où le point doit être inscrit ; on utilise alors une instruction PRINT pour imprimer un motif à cette position. Celui utilisé est le caractère 143. Si vous examinez, dans le manuel CPC 464, le jeu de caractères (appendice III) vous remarquez que c'est simplement une case colorée par le stylo utilisé. Chaque point demande deux de ces cases et chacune nécessite des instructions LOCATE et PRINT séparées.

Ce premier sous-programme dessine donc le point milieu. Le suivant inscrit les points en haut à droite et en bas à gauche, le troisième, les deux autres coins. Le dernier enfin dessine les points milieux latéraux. En appelant les sous-programmes désirés lors d'instructions IF ... THEN, on peut alors dessiner le dé correctement. Par exemple le deuxième et le troisième sous-programmes donnent un quatre, les trois derniers donnent un six. Des jeux de sous-programmes différents sont utilisés pour les deux dés.

Il est important de noter que l'instruction LOCATE n'utilise pas les coordonnées usuelles de l'écran. En fait chaque fenêtre a sa propre origine 1-1 des coordonnées, elle se trouve dans le coin en haut à gauche. Ainsi le centre du dé de gauche n'est pas en 6-11 et 6-12 mais en 4-7 et 4-8. Bien entendu, puisque les coordonnées utilisées dans les sous-programmes pour le dé gauche sont exactement les mêmes que celles utilisées pour le dé droit, la seule différence entre ces sous-programmes est le numéro d'identification utilisé dans les instructions LOCATE et PRINT.

Les lignes 330 à 360 servent à faire boucler le programme un certain nombre de fois pour obtenir le roulement des dés. La variable x prend aléatoirement les valeurs 0, 1 et 2 à chaque bouclage du programme, et les dés roulent tant qu'elle ne vaut pas 1. En moyenne les dés roulent donc trois fois, mais cela peut varier de 1 à 12 à peu près. Les chiffres anciens doivent être effacés avant d'inscrire les suivants ; cela se fait rapidement grâce à une instruction CLS pour

chaque fenêtre. La ligne 370 interrompt le programme, jusqu'à l'instant où l'on appuie sur la touche d'intervalle ; les fenêtres sont alors effacées et le programme revient en 180 pour un nouveau roulement des dés.

LISTING 16

```
20 MODE 0
30 INK 0,2
40 BORDER 18
50 WINDOW#1,3,9,5,18
60 WINDOW#2,12,18,5,18
70 INK 2,25
80 INK 3,15
90 INK 4,8
100 INK 5,6
110 PAPER#1,2
120 PAPER#2,4
130 PEN#1,3
140 PEN#2,5
150 CLS#1
160 CLS#2
170 RANDOMIZE TIME
180 l = INT (RND*6+1)
190 r = INT (RND*6+1)
200 SOUND 1,1,2,7,0,0,2
210 IF l = 1 THEN GOSUB 400
220 IF l = 2 THEN GOSUB 430
230 IF l = 3 THEN GOSUB 400:GOSUB 430
240 IF l = 4 THEN GOSUB 430:GOSUB 480
250 IF l = 5 THEN GOSUB 400:GOSUB 430
:GOSUB 480
260 IF l = 6 THEN GOSUB 430:GOSUB 480
:GOSUB 530
270 IF r = 1 THEN GOSUB 580
280 IF r = 2 THEN GOSUB 610
290 IF r = 3 THEN GOSUB 580:GOSUB 610
300 IF r = 4 THEN GOSUB 610:GOSUB 660
```

```
310 IF r = 5 THEN GOSUB 580:GOSUB 610:GOSUB 660
320 IF r = 6 THEN GOSUB 610:GOSUB 660:GOSUB 710
330 x = INT (RND*3)
340 IF x = 1 THEN 370
350 CLS#1:CLS#2
360 GOTO 180
370 a$ = INKEY$:IF a$ <> CHR$<32> THEN GOTO
    370
380 CLS#1:CLS#2
390 GOTO 180
400 LOCATE#1,4,7:PRINT#1, CHR$(143)
410 LOCATE#1,4,8:PRINT#1, CHR$(143)
420 RETURN
430 LOCATE#1,6,3:PRINT#1, CHR$(143)
440 LOCATE#1,6,4:PRINT#1, CHR$(143)
450 LOCATE#1,2,11:PRINT#1, CHR$(143)
460 LOCATE#1,2,12:PRINT#1, CHR$(143)
470 RETURN
480 LOCATE#1,2,3:PRINT#1, CHR$(143)
490 LOCATE#1,2,4:PRINT#1, CHR$(143)
500 LOCATE#1,6,11:PRINT#1, CHR$(143)
510 LOCATE#1,6,12:PRINT#1, CHR$(143)
520 RETURN
530 LOCATE#1,2,7:PRINT#1, CHR$(143)
540 LOCATE#1,2,8:PRINT#1, CHR$(143)
550 LOCATE#1,6,7:PRINT#1, CHR$(143)
560 LOCATE#1,6,8:PRINT#1, CHR$(143)
570 RETURN
580 LOCATE#2,4,7:PRINT#2, CHR$(143)
590 LOCATE#2,4,8:PRINT#2, CHR$(143)
600 RETURN
610 LOCATE#2,6,3:PRINT#2, CHR$(143)
620 LOCATE#2,6,4:PRINT#2, CHR$(143)
630 LOCATE#2,2,11:PRINT#2, CHR$(143)
640 LOCATE#2,2,12:PRINT#2, CHR$(143)
650 RETURN
660 LOCATE#2,2,3:PRINT#2, CHR$(143)
670 LOCATE#2,2,4:PRINT#2, CHR$(143)
680 LOCATE#2,6,11:PRINT#2, CHR$(143)
```

```
690 LOCATE#2,6,12:PRINT#2, CHR$(143)
700 RETURN
710 LOCATE#2,2,7:PRINT#2, CHR$(143)
720 LOCATE#2,2,8:PRINT#2, CHR$(143)
730 LOCATE#2,6,7:PRINT#2, CHR$(143)
740 LOCATE#2,6,8:PRINT#2, CHR$(143)
750 RETURN
```

WINDOW SWAP *(ECHANGE DE FENÊTRES)*

WINDOW SWAP est une instruction utilisée pour permuter les numéros d'identification de deux fenêtres. Par exemple pour échanger les données de la fenêtre 3 à la fenêtre 4, on utilise : WINDOW SWAP 3,4.

Cette commande est utilisable pour échanger des informations de l'écran principal (de numéro 0) à une fenêtre. Par exemple WINDOW SWAP 0,4 transférera les données de l'écran principal dans la fenêtre 4. Il est possible de modifier le programme de dés en utilisant cette instruction et un seul jeu de sous-programmes. Il est instructif d'essayer cette modification.

Chapitre 7

GRAPHISME 2 — ANIMATION

Le principe de l'animation est simple. On imprime un caractère sur l'écran, il est ensuite effacé, puis réimprimé à une place adjacente. En répétant cette opération avec un rythme approprié, le caractère semble bouger sur l'écran.

Il est également possible d'animer un objet constitué de plusieurs caractères, c'est assez facile si les déplacements ne sont qu'horizontaux. Le Listing 17 illustre ceci, il déplace un seau, constitué de deux caractères graphiques et de deux tirets de part et d'autre du seau en bas de l'écran. En mettant un espace (CHR$(32)) de chaque côté du motif, il s'efface directement lorsqu'il se déplace. Ce déplacement est contrôlé à partir du clavier : par la touche gauche et la touche \ (déplacement droit). Ces touches sont contrôlées par la fonction INKEY utilisée aux lignes 60 et 70. INKEY vaut – 1 lorsque la touche spécifiée n'est pas enfoncée ; lorsqu'elle l'est, cette valeur dépend de l'emploi ou non des touches *control* et *shift*. Il est facile de vérifier si la touche est enfoncée ou non en utilisant IF NOT que l'on a traité dans le chapitre 3.

La ligne 95 demande des explications. Quand le programme est exécuté, on voit parfois une image double du seau qui se déplace. CALL αBD19 appelle un sous-programme interne qui provoque une interruption jusqu'au commencement d'un nouveau cycle vidéo. Nous l'avons essayé pour éliminer cette image double, mais vous conviendrez avec nous que le résultat n'est pas évident.

Maintenant que vous savez déplacer le seau, vous pouvez le remplir d'objets. Voyons comment opérer des déplacements verticaux ; nous pourrons alors combiner ces deux mouvements et les utiliser pour divers jeux.

LISTING 17

```
2Ø REM deplacement d'un seau
25 CLS
3Ø bucket$=CHR$(32)+CHR$(2Ø5)+"--
"+CHR$(2Ø4)+CHR$(32)
4Ø bx=17:by=24
5Ø WHILE 1
6Ø IF NOT INKEY(71) THEN bx=bx-1
7Ø IF NOT INKEY(22) THEN bx=bx+1
8Ø IF bx<1 THEN bx=1
9Ø IF bx>34 THEN bx=34
95 CALL &BD19
1ØØ LOCATE bx,by:PRINT bucket$
11Ø WEND
```

Le listing 18 permet de déplacer verticalement un motif en forme de cœur (CHR $ (228)).

La ligne 40 déclenche le générateur de nombres aléatoires. La ligne 45 donne des valeurs initiales à deux variables utilisées pour enregistrer la position du motif à effacer.

La descente est effectuée par une boucle FOR... NEXT des lignes 60 à 100. La variable de contrôle sert de coordonnée verticale. La ligne 70 imprime le cœur et la ligne 80 l'efface. La ligne 90 remet à jour les variables odx et ody.

LISTING 18

```
2Ø REM déplacement vertical
3Ø CLS
4Ø RANDOMIZE TIME
45 odx=1:ody=1
5Ø LET dx=INT(RND*3Ø+5)
6Ø FOR dy=1 TO 23
7Ø LOCATE dx,dy:PRINT CHR$(228)
8Ø LOCATE odx,ody:PRINT CHR$(32)
9Ø odx=dx:ody=dy
1ØØ NEXT dy
11Ø GOTO 5Ø
```

Le listing 19 est un programme de jeu qui rassemble les deux opérations précédentes. La première partie du programme des lignes 20 à 70 réunit les éléments qui doivent être définis d'abord, c'est-à-dire, par exemple, la création (d'un seau) et des inscriptions sur l'écran. La seconde partie des lignes 90 à 100 crée une séquence réalisée pour chaque projectile. La ligne 100 sélectionne le caractère à imprimer, entre le cœur et la flèche dirigée vers le bas. Le but du jeu est d'attraper les cœurs (10 points), et d'éviter les flèches (pénalité : le score est divisé par deux). La ligne 110 détermine la position horizontale, comme dans le Listing 18. La ligne 120 appelle le sous-programme qui permet de jouer. On peut remarquer qu'il est construit à partir des Listings 17 et 18 avec quelques instructions supplémentaires comme PEN et SOUND.

Lorsque chaque projectile a atteint le bas de l'écran, la ligne 130 compare la coordonnée horizontale du projectile et celle du seau. Si elles sont semblables (le projectile doit être entouré d'espaces pour être valable), le sous-programme débutant en 2000 est appelé. Ce sous-programme utilise la variable C qui détermine la couleur du projectile et permet de changer le score (en ligne 2010 et 2020 avec des effets sonores appropriés). Ce sous-programme imprime aussi le score sur l'écran.

Une partie comporte 20 projectiles. Lorsqu'elle est terminée, la ligne 150 appelle le sous-programme débutant en ligne 3000, qui permet soit une autre partie, soit la fin du jeu. La ligne 3015 est intéressante. On utilise la fonction INKEY pour contrôler l'enfoncement d'une touche, mais elle ne peut pas supprimer un caractère de commande. Cela signifie que toutes les touches utiles au déplacement du seau doivent être enregistrées. La ligne 3015 élimine ces touches de sorte que le programme ne chute pas en 3030. La ligne 3060 permet de rester à l'intérieur de ce sous-programme si une de ces touches est enfoncée par mégarde, ce qui est assez fréquent. Ce sous-programme n'interprète pas la réponse du joueur.

La ligne 160 décide de l'action à exécuter. Si la touche y est enfoncée, une nouvelle partie commence. Si une autre

touche est sélectionnée, la ligne 170 réinitialise les couleurs, de mode de l'écran, et la partie s'interrompt.

Le sous-programme des lignes 4000 à 4050 ne sert qu'à inscrire le plus haut score enregistré depuis le commencement du programme.

Le dernier sous-programme, qui commence à la ligne 5000, crée une pause. Cette interruption prend fin lorsqu'une touche est enfoncée, cela donne le temps d'être prêt. Ce sous-programme est appelé par la ligne 85.

Ce jeu a été conçu avec les couleurs initiales du mode 1. Il se peut que les flèches soient assez difficiles à voir (elles sont jaune clair). Si c'est le cas, il est facile d'ajouter une ligne au début du programme (la ligne 45 par exemple) pour changer la couleur de PEN 1.

L'emploi de coordonnées dans l'animation rend ce procédé un peu saccadé, mais il est rapide, passionnant, et très utile pour les jeux simples de ce type.

LISTING 19

```
2Ø REM jeu du seau
3Ø MODE 1
4Ø BORDER 18:PAPER 2:CLS
5Ø bucket$=CHR$(32)+CHR$(2Ø5)+"--"+CHR$(2Ø4)+
CHR$(32)
55 bx=17:by=24
6Ø LOCATE 1,1:PEN 0:PRINT "SCORE"
7Ø LOCATE 34,1:PRINT "HISCORE"
8Ø RANDOMIZE TIME
85 GOSUB 5ØØØ:REM Procédure "appuyer sur une
touche"
9Ø FOR bombs=1 TO 2Ø:REM création des bombes
1ØØ IF RND>Ø.5 THEN bomb$=CHR$(228):c=3:ELSE
bomb$=CHR$(241):c=1
11Ø dx=INT(RND*25)+7
12Ø GOSUB 1ØØØ:REM procédure de jeu
13Ø IF (dx=bx+2) OR (dx=bx+3) THEN GOSUB
    2ØØØ:REM score
```

```
135 LOCATE odx,ody:PRINT CHR$ (32)
140 NEXT bombs
150 GOSUB 3000:REM une autre partie ?
155 GOSUB 4000:REM hiscore
160 IF LOWER$(ans$)=“y” THEN 85
170 CALL &BBFF:PAPER 0:PEN 1:CLS
180 END
1000 REM procédure de jeu
1010 odx=7:ody=1
1020 FOR dy=1 TO 23
1030 LOCATE odx,ody:PRINT CHR$(32)
1040 dx,dy:PEN c:PRINT bomb$
1050 odx=dx:ody=dy
1055  SOUND 1,dy*50,5
1060 IF NOT INKEY(71) THEN bx=bx-1
1070 IF NOT INKEY(22) THEN bx=bx+1
1080 IF bx<1 THEN bx=1
1090 IF bx>34 THEN bx=34
1100 LOCATE bx,by:PEN 0:PRINT bucket$
1110 NEXT dy
1120 RETURN
2000 REM score
2010 IF c=3 THEN pts=pts+10:SOUND 1,50,50
2020 IF c=1 THEN pts=CINT(pts/2):SOUND 1,1000,100
2030 LOCATE 1,2:PEN 0:PRINT pts
2040 RETURN
3000 LOCATE 15,10
3010 PEN 3
3015 WHILE INKEY$<>“”:WEND
3020 PRINT “Une autre partie ? (y/n)”
3030 ans$=INKEY$:IF ans$=“” THEN 3030
3040 LOCATE 15,10
3050 PRINT SPACE$(18)
3060 IF LOWER$(ans$)=“z” OR ans$=“\” THEN 3000
3070 RETURN
4000 IF pts>hiscore THEN hiscore=pts
4010 LOCATE 34,2
4020 PEN 0
4030 PRINT hiscore
```

```
4Ø4Ø pts=Ø
4Ø5Ø RETURN
5ØØØ REM sous-programme de départ
5Ø1Ø LOCATE 1Ø,1Ø
5Ø2Ø PEN 3
5Ø3Ø WHILE INKEY$<>"":WEND
5Ø4Ø PRINT "appuyez sur une touche pour commencer"
5Ø5Ø a$=INKEY$:IF a$="" THEN 5Ø5Ø
5Ø6Ø LOCATE 1Ø,1Ø
5Ø7Ø PRINT SPACE$(22)
5Ø8Ø RETURN
```

GRAPHISME ET CURSEUR DE TEXTE

Lorsqu'on veut faire de l'animation d'une manière plus continue, il est utile de mettre des caractères de texte à l'endroit du curseur de texte. Les caractères peuvent être déplacés sur l'écran, ce déplacement semble plus continu puisqu'une seule partie du caractère bouge à la fois.

La commande appropriée est TAG. Pour redonner un emploi normal au curseur de texte, on utilise TAGOFF ; Si le Basic revient en mode commande, c'est automatique.

Lorsque TAG est appliqué, la couleur des caractères imprimés ne dépend pas des instructions PEN. C'est-à-dire qu'ils seront inscrits dans la couleur spécifiée par les commandes DRAW et PLOT (ou bien celle de *pen 1* par défaut). De même, la couleur de l'écran est celle que l'on a déterminée (ou *pen 0* par défaut).

L'Amstrad a un nombre de modes de couleur tel, qu'il permet à la création de graphiques et à la commande TAG de réagir à ce qui est déjà sur l'écran. Par défaut on utilise le mode de couleur 0 ; il impose la couleur de l'inscription et le fond de l'écran indépendamment de ce qui précède. En mode 1, la couleur qui apparaît est celle que l'on définit, plus la couleur déjà en place, (commande XOR). Le mode 2 utilise AND *(et)* et le mode 3 OR *(ou)*.

Pour l'animation, XOR est le plus souvent utilisé, ceci pour inscrire la même chose deux fois à la même place ; XOR

rétablit les couleurs initiales, un caractère peut donc bouger sur un fond multicolore sans avoir à changer ce fond. Par suite, si la couleur des graphiques du fond de l'écran est définie par *pen 0,* la couleur du papier sur lequel est écrit le caractère est transparent puisque *(nombre* XOR 0*)* = *nombre.*

En utilisant XOR, il est également possible, d'une manière simple, de déplacer un caractère en le faisant passer devant des objets, sur l'écran ou bien derrière d'autres. Le Listing 20 illustre cette opération. Il dessine un désert de sable jaune, avec un ciel bleu, et deux pyramides rouges. Un petit tank noir se déplace derrière la première pyramide et devant la seconde.

Pour faire ceci, nous devons déclarer, un certain nombre de couleurs de stylos. Lorsque l'on réalise ce genre d'effets graphiques, il est plus facile d'utiliser un numéro de stylo de 8 ou plus pour le caractère, et un numéro inférieur à 8 pour le fond. Dans notre exemple, *pen 8* est utilisé pour le tank.

Pen 3 sert au désert. Ce stylo est jaune, comme le définit la commande INK à la ligne 30. 3 XOR 8 vaut 11, comme le tank est noir, *pen 11* doit être noir, ce qui est fait en ligne 50.

Nous devons prendre des stylos différents pour les deux pyramides. *Pen 5* est utilisé pour la première, et *pen 1* pour la seconde. Ces deux stylos sont rouges comme l'indiquent les lignes 60 et 70 ; 1 XOR 8 vaut 9. Comme nous voulons que le tank passe devant la première pyramide, *pen 9* doit être noir (ligne 80).

Notons qu'il n'est pas nécessaire de réattribuer une couleur à *pen 8,* puisqu'on ne l'utilise jamais !

Les lignes 80 et 90 colorent l'écran en jaune. Les lignes 100 à 120 créent le ciel grâce à une fenêtre, puis lui donnent une couleur bleue. Les lignes 140 et 150 dessinent les pyramides en appelant le sous-programme qui commence en 1000. Celui-ci se sert d'une boucle FOR... NEXT pour remplir de lignes les formes triangulaires.

Il n'y a pas d'instruction Basic pour spécifier le mode de couleur. Cela se fait en envoyant des codes de contrôle au VDU. La ligne 180 illustre ceci ; CHR$(23) est le code à utiliser pour sélectionner le mode de couleur, il est

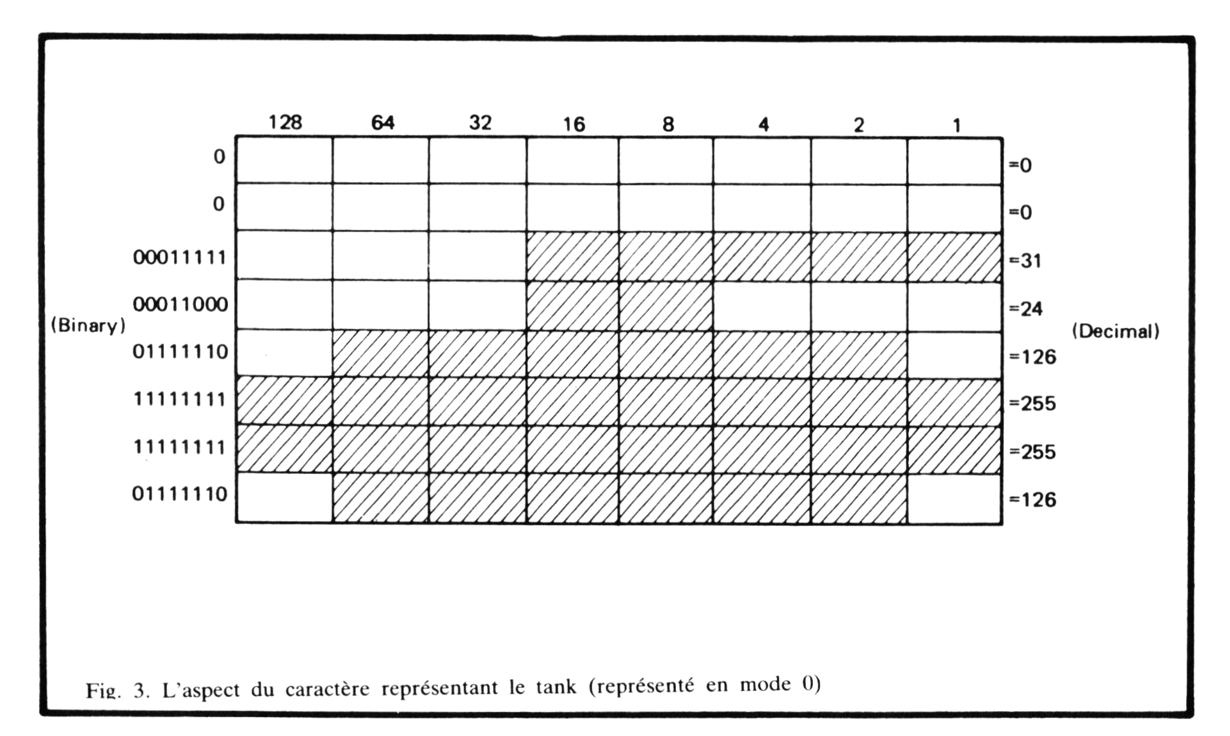


Fig. 3. L'aspect du caractère représentant le tank (représenté en mode 0)

directement suivi par le mode désiré. On doit aussi utiliser un point-virgule, ou le signe + (assemblage) entre les deux caractères. Tout autre signe de séparation (comme la virgule) n'opérera pas. La ligne 180 réinitialise le mode en fin de programme. Le tank bouge sur l'écran grâce au sous-programme qui débute en ligne 2000. Le principe est semblable à celui utilisé par les projectiles du dernier programme, avec l'emploi de variables pour mémoriser la position ancienne. Dans notre exemple, seule la coordonnée x change ; nous avons donc x ou ox mais pas oy. L'effacement se fait en réimprimant le caractère, plutôt qu'en imprimant un espace par dessus. Les lignes 2050 et 2070 réalisent l'impression du tank. La ligne 2020 détermine le point de départ en un point extérieur à l'écran. On emploie & BD19 pour améliorer sensiblement l'animation dans ce programme. La ligne 2085 constitue simplement une courte pause pour contrôler la vitesse du mouvement.

Une fois le programme fini, la ligne 190 attend que l'on appuie sur une touche, la ligne 200 remet alors le mode à 1 et les couleurs à leurs valeurs initiales. Le tank, CHR $ (254) est un caractère redéfini (ligne 130). La figure 3 montre comment les nombres spécifiés dans l'instruction SYMBOL sont agencés (le premier, 254, étant le code du caractère à redéfinir). Bien sûr il n'est pas nécessaire de traduire ces nombres en décimaux puisque l'Amstrad accepte les nombres binaires en les faisant précéder de &X.

LISTING 2Ø

```
2Ø MODE Ø
3Ø INK 3,12
4Ø INK 9,Ø
5Ø INK 11,Ø
6Ø INK 1,3
7Ø INK 5,3
8Ø INK 13,3
9Ø PAPER 3:CLS
1ØØ WINDOW #1,1,2Ø,1,1Ø
```

```
110 PAPER #1,6
120 CLS#1
130 SYMBOL 254,0,0,31,24,126,255,25,5,5,126
140 c=5:x=250:y=150:GOSUB 1000
150 c=1:x=450:y=200:GOSUB 1000
160 PRINT CHR$(23);CHR$(1);
170 GOSUB 2000
180 PRINT CHR$(23);CHR$(0);
190 WHILE INKEY$=« »:WEND
200 MODE 1:CALL &BBFF:PAPER 0
210 END
1000 REM dessin des pyramides
1010 w=200
1020 FOR v=1 TO 100
1030 MOVE x-w/2,y+v
1040 DRAWR w,0,c
1050 w=w-2
1060 NEXT v
1070 RETURN
2000 TAG
2010 PLOTR 0,0,0
2020 y=212:ox=-64:REM point de départ du tank
2030 FOR x=0 TO 640 STEP 4
2040 MOVE ox,y
2045 CALL &BD19
2050 PRINT CHR$(254);
2060 MOVE x,y
2065 CALL &BD19
2070 PRINT CHR$(254);
2080 ox=x
2085 FOR d=1 TO 100:NEXT
2090 NEXT x
2100 TAGOFF
2110 RETURN
```

MANETTES ET TEST

Lorsque les déplacements sur l'écran doivent intervenir dans quatre directions, les manettes sont préférables au clavier. Le Listing 21 est la version informatisée du jeu dans lequel vous devez déplacer un anneau le long d'un fil penché sans que l'anneau touche le fil. Les choses sont quelque peu différentes cependant, puisque vous avez à tracer une ligne jaune entre deux lignes rouges parallèles sans les toucher. Ce programme illustre la fonction TEST, qui donne la nature de l'encre en un point spécifié sur l'écran.

Le trajet que suivent les lignes parallèles est établi au hasard. Ce jeu est plus distrayant en compétition, et comme les trajets peuvent être de difficultés différentes, le programme a été conçu de sorte qu'un trajet peut être répété le nombre de fois que l'on veut.

Le sous-programme qui débute à la ligne 1000 donne les coordonnées des lignes parallèles à l'aide de RND (ligne 1020), et les mémorise dans un tableau. Le sous-programme qui commence en 2000 dessine l'écran avec une frontière rouge (lignes 2020 à 2050), et les deux lignes parallèles (2060 à 2160). La ligne 2170 donne le point de départ de la ligne jaune.

La bouche WHILE ... WEND des lignes 80 à 110 appelle régulièrement les sous-programmes qui commencent en 3000 et 4000 jusqu'à ce que la ligne jaune atteigne le côté droit de l'écran. Le sous-programme qui démarre en 3000 effectue l'essentiel du jeu proprement dit. Les lignes 3020 et 3030 enregistrent les données à partir des manettes, et positionnent les coordonnées px et py, utilisées pour le dessin comme demandé. On ne peut se déplacer en diagonale. Remarquons que même si la plupart des manettes ont les mêmes types de prises, il existe une différence d'opinion entre les constructeurs quant à la manière de les utiliser. Si vous trouvez que vos manettes opèrent dans le mauvais sens pour ce programme, vous pouvez intervertir les signes + et – des lignes 3020 et 3030, ou bien retourner votre manette !

La ligne 3040 vérifie si le point à imprimer n'est pas rouge *(ink 3)*. S'il l'est, px et py prennent leur valeur précédente

(voir aussi la ligne 3010), et vous êtez pénalisé d'une seconde ; un bourdonnement est alors produit. Sinon le nouveau point est imprimé (ligne 3050).

Le sous-programme qui commence en 4000 inscrit le temps passé sur l'écran, à l'aide de la fonction TIME. La variable t prend la valeur de l'horloge interne en ligne 70. La ligne 4020 calcule le temps écoulé en soustrayant t à la valeur de TIME et divise ce nombre par 300 (puisque l'horloge interne est incrémentée tous les 1/300e de seconde). On voit aussi comment, en ajoutant 300 à la valeur de t, on peut obtenir une seconde de pénalité lorsque l'on touche les lignes (ligne 3040).

La ligne 4020 utilise PRINT USING. C'est une chaîne de caractères qui contrôle le format de l'inscription sur l'écran. Dans notre exemple, le contrôle est imprimé dans trois cases à gauche du point décimal et dans deux à droite. Il y a plusieurs sortes d'indications PRINT, à la fois pour les chaînes et les nombres, et la meilleure manière de les retenir est de les expérimenter.

Lorsque la ligne jaune est dessinée, le sous-programme qui démarre en ligne 5000 vous demande si vous voulez reparcourir l'écran. Si vous répondez y, le programme retourne en ligne 60 pour redessiner le parcours.

Si vous répondez n, vous pouvez choisir un nouveau trajet ; si vous répondez alors y, le programme retourne en ligne 50. Sinon il s'arrête, l'ordinateur revient en mode 1, et les couleurs reprennent leurs valeurs initiales.

Une caractéristique intéressante de l'Amstrad réside dans le fait que le déplacement des manettes correspond effectivement à des caractères du clavier. Ces caractères enregistrés ne doivent pas être les mêmes que ceux utilisés dans les lignes 5030 et 6030, c'est pourquoi ces lignes n'ont employé que les caractères y et n.

LISTING 21

```
2Ø REM * JEU DE LIGNES *
3Ø MODE Ø
4Ø DIM y%<8>
5Ø GOSUB 1ØØØ:REM création du trajet
6Ø GOSUB 2ØØØ:REM tracé de l'écran
7Ø t=TIME
8Ø WHILE px%<63Ø
9Ø GOSUB 3ØØØ:REM déplacement
1ØØ GOSUB 4ØØØ:REM inscription du temps
11Ø WEND
12Ø GOSUB 5ØØØ:REM "une autre tentative?"
13Ø IF LOWER$(ans$)="y" THEN GOTO 6Ø
14Ø GOSUB 6ØØØ:REM "un nouveau trajet?"
15Ø IF LOWER$(ans$)="y" THEN GOTO 5Ø
16Ø MODE 1:CALL &BBFF:PAPER Ø:PEN 1
17Ø END
1ØØØ REM coordonnées du trajet
1Ø1Ø FOR points=Ø T0 8
1Ø2Ø y%(points)=RND*2ØØ+1ØØ
1Ø3Ø NEXT points
1Ø4Ø RETURN
2ØØØ REM tracé de l'écran
2ØØ5 MOVE Ø,Ø
2Ø1Ø PAPER 8:BORDER 3:CLS
2Ø2Ø DRAWR Ø,399,3
2Ø3Ø DRAWR 639,Ø
2Ø4Ø DRAWR Ø,-399
2Ø5Ø DRAWR -639,Ø
2Ø6Ø MOVE Ø,y%(Ø)+2Ø
2Ø65 points=Ø
2Ø7Ø FOR x%=79 T0 639 STEP:8Ø
2Ø8Ø DRAW x%,y%(points)+2Ø
2Ø9Ø points=points+1
21ØØ NEXT x%
211Ø points=Ø
212Ø MOVE Ø,y%(Ø)-2Ø
213Ø FOR x%=79 TO 639 STEP 8Ø
```

```
214Ø DRAW x%,y%(points)−2Ø
215Ø points=points+1
216Ø NEXT x%
217Ø px%=1Ø:py%=y%(Ø)
218Ø RETURN
3ØØØ REM déplacement
3Ø1Ø a%=px%:d%=py%
3Ø2Ø px%=px%+4*(JOY(Ø)=8)−4*(JOY(Ø)=4)
3Ø3Ø py%=py%+2*(JOY(Ø)=1)−2*(JOY(Ø)=2)
3Ø4Ø IF TEST (px%,py%)=3 THEN px%=a%:py%=d%,
     t=t−3ØØ:SOUND 1,1ØØØ,2Ø
3Ø5Ø PLOT px%,py%,1
3Ø6Ø RETURN
4ØØØ REM inscription du temps écoulé
4Ø1Ø LOCATE 2,2
4Ø2Ø PRINT USING "###,##";(TIME−t)/3ØØ
4Ø3Ø RETURN
5ØØØ REM sous programme "nouvelle tentative"
5Ø1Ø LOCATE 2,4
5Ø2Ø PRINT "une nouvelle tentative? (y/n)"
5Ø3Ø ans$=LOWER$(INKEY$):IF ans$<>"y" AND
ans$<>"n" THEN 5Ø3Ø
5Ø4Ø LOCATE 2,4
5Ø5Ø PRINT SPACE$(16)
5Ø6Ø RETURN
6ØØØ REM sous programme "nouveau tracé"
6Ø1Ø LOCATE 2,4
6Ø2Ø PRINT "un nouveau trajet? (y/n)"
6Ø3Ø ans$=LOWER$(INKEY$):IF ans$<>"y" AND
ans$<>"n" THEN 6Ø3Ø
6Ø4Ø LOCATE 2,4
6Ø5Ø PRINT SPACE$(16)
6Ø6Ø RETURN
```

Chapitre 8

BINAIRE ET HEXADÉCIMAL

Une grande partie de la programmation peut être effectuée sans savoir comment l'ordinateur travaille, et même sans comprendre quelques principes de base, surtout lorsque l'on utilise un haut niveau de langage, comme c'est le cas de Locomotive Basic de l'Amstrad CPC 464. Cependant, même si ce langage est sophistiqué, certains aspects de la programmation ne peuvent être entièrement compris que si l'on connaît les opérations fondamentales que réalise un ordinateur. Le système de numérotation binaire constitue peut-être le meilleur exemple. Les ordinateurs ne travaillent pas directement dans notre système de numérotation ordinaire, mais en binaire, sous une forme ou une autre.

Le système que nous utilisons est appelé communément : système décimal. Il est bien sûr, basé sur le nombre 10. Il y a 10 éléments numérotés de 0 à 9. Ce système ne s'adapte pas très bien aux circuits électroniques, en ce sens qu'il est difficile de concevoir pratiquement un circuit dans lequel la sortie peut prendre 10 niveaux de tension différents pour pouvoir représenter un nombre décimal. C'est pourquoi on utilise le système de numérotation binaire.

Ce système est basé sur le nombre 2 au lieu de 10, le plus grand élément est 1 au lieu de 9. Si l'on considère un nombre décimal comme 238, 8 représente huit unités (10 à la puissance 0), 3 représente trois dizaines (10 à la puissance 1 et 2 représente deux centaines (10 à la puissance 2, ou 10 au carré). Il en est de même pour un nombre binaire comme 1 101. On travaille de droite à gauche. 1 représente les unités (2 à la puissance 0), 0 représente le nombre de deux (2 à la puissance 1), le 1 représente le nombre de quatre (2 à la puissance 2), le dernier 1 représente le nombre de huit (2 à la puissance 3). 1 101 en binaire est équivalent à 13 en décimal (1+0+4+8=13).

La table donnée ci-dessous indique les nombres représentés par chaque élément d'un nombre de 16 bits lorsque

ceux-ci sont en position haute (1). Bien sûr, un élément correspond toujours à 0 lorsqu'il est en position basse (0).

BIT	0	1	2	3	4	5	6	7	8
	1	2	4	8	16	32	64	128	256

BIT	9	10	11	12	13	14	15
	512	1024	2048	4096	8192	16384	32768

Les éléments constituant les nombres du système binaire sont appelés bits, un groupe de 8 bits est appelé octet.

En utilisant 16 bits, on peut représenter en binaire n'importe quel entier compris entre 0 et 65 535 ; si l'on en prend 8, cet entier est compris entre 0 et 255, voici le principal défaut du système binaire. Des nombres de valeurs peu élevées utilisent beaucoup de bits, mais en dépit de cet inconvénient, l'adaptation des circuits électroniques au binaire fait de ce système le seul utilisable actuellement.

L'addition de deux nombres binaires est une opération directe, qui est en fait plus simple que l'addition décimale. Voici un exemple :

premier nombre	1	1	1	1	0	0	0	0
deuxième nombre	0	1	0	1	0	1	0	1
somme	10	1	0	0	0	1	0	1

Comme pour l'addition décimale, on commence par la colonne des unités, puis progressivement vers la gauche, jusqu'à la dernière colonne. Dans notre exemple, il y a un 1 et un 0 dans la colonne des unités qui donnent un total de 1 comme unité du nombre somme. La colonne suivante comporte deux zéros qui donnent 0, de même l'addition des deux colonnes suivantes se fait directement. Dans la cinquième, on doit ajouter deux 1 qui donnent un total de 2. Bien sûr 2 n'existe pas en binaire et doit être considéré comme 10 (une fois 2 et 0 unités) ; 2 est traité de la même manière que 10 en décimal. La réponse est donc 0, et 1 est

retenu pour la prochaine colonne. La sixième colonne donne encore un total de 10.

0 est placé dans le résultat, et 1 est retenu. Le septième donne en décimal un total de 3, mais en système binaire on doit prendre 11 (une fois 2 plus 1 unité). C'est pourquoi 1 est mis dans la réponse, et un autre 1 est retenu pour la colonne suivante. Le total de la huitième colonne est 10, et puisqu'il n'y a pas d'autres colonnes à additionner, ces deux bits apparaissent dans le résultat.

Une explication détaillée de la façon dont l'ordinateur réalise des opérations complexes en binaire serait ici hors de propos.

Si vous avez compris comment des nombres peuvent être représentés par des circuits électroniques, à l'aide du système binaire, vous pourrez comprendre et utiliser les fonctions du CPC 464 qui demandent une connaissance du binaire.

BIN$

Une particularité utile du CPC 464, que peu de micro-ordinateurs possèdent, est la fonction BIN$. Cette fonction prend la forme suivante : le nombre spécifié (décimal) est converti en binaire. Par exemple :

```
PRINT BIN$ (8)
```

vous donnera 1000.

Il existe une version un peu plus délicate de cette commande ; le premier nombre est à convertir en binaire, le deuxième spécifie le nombre de bits utilisés. Par exemple :

```
PRINT BIN$ (8,12)
```

donnera 000000001000

En d'autres termes, elle convertit 8 en binaire ; le résultat étant donné à l'aide de 12 bits, les zéros qui se trouvent devant ne sont pas supprimés. Il faut noter que si l'on donne

un nombre de bits inférieur à celui qui est nécessaire à la conversion, il n'y a pas de suppression.

Ainsi la commande

PRINT BIN$ (8,2)

inscrira

1000

et non pas

00

Le CPC 464 accepte les nombres binaires, ils doivent être précédés de &x pour indiquer à l'ordinateur qu'il s'agit de nombres binaires et non pas décimaux. L'ordinateur peut aussi donner la traduction décimale d'un nombre binaire. Par exemple :

PRINT &x 1000

imprimera 8 sur l'écran.

Le nombre le plus grand que l'on peut traiter en binaire est 65 535 ou 1111111111111111 (soit 16 bits).

On peut utiliser des nombres négatifs, mais la manière de les utiliser dépasse le propos de ce livre.

HEXADÉCIMAL

Puisque nous étudions les systèmes de numérotation, il serait certainement intéressant d'en traiter un que vous rencontrerez inévitablement, et de plus, fréquemment ; le système hexadécimal. Un inconvénient du système binaire tient dans le fait qu'il demande un grand nombre de bits, chaque bit valant 0 ou 1, ce qui le rend parfois peu pratique. D'un autre côté, les nombres binaires donnent une représentation graphique de l'état de chaque bit des registres du microprocesseur, et c'est un élément souvent très important. Les nombres décimaux sont plus faciles à utiliser parce qu'ils sont plus courts et de forme familière. Cependant, il n'est pas très rapide ni très facile de convertir des nombres

décimaux en binaire, particulièrement lorsque ces nombres sont grands, et ce n'est pas commode lorsqu'il est nécessaire de visualiser cette opération par un procédé bit-à-bit.

Le système hexadécimal réunit les avantages de ces deux systèmes, puisqu'il ne faut qu'un petit nombre d'éléments pour représenter des nombres assez élevés ; il est même, dans cette optique, plus performant que le système décimal. D'autre part, il est aisé à convertir en binaire, et il est facile à utiliser, lorsqu'on l'opère au niveau des bits. Le système hexadécimal est basé sur le nombre 16, il comporte donc seize éléments. Bien évidemment les nombres que nous utilisons dans le décimal ne sont pas valables en hexadécimal. Puisqu'ils ne sont pas assez nombreux, on résoud ce problème en ajoutant à ces dix nombres, six lettres de l'alphabet d'où le nom donné à ce système. La table qui suit vous aidera à comprendre la manière d'opérer du système hexadécimal.

Décimal	*Hexadécimal*	*Binaire*
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	00010000
17	11	00010001
163	A3	10100011

Ce qui rend l'hexadécimal si pratique, c'est sa très grande aptitude à être converti en binaire, puisque chaque élément hexadécimal est représenté par quatre bits. Considérons le nombre hexadécimal A3 dans la table. En binaire A représente 1010 et 3, 0011. Ainsi A3 représente 10100011 en binaire. Peut-être pensez-vous qu'il faut mémoriser les quatre bits représentant les seize éléments de l'hexadécimal ? En fait une petite opération mentale arithmétique est suffisante pour la conversion.

Les éléments d'un nombre hexadécimal représentent, de gauche à droite, le nombre d'unités, le nombre de fois 16, 256 et 4096. Vous ne pouvez malheureusement pas employer des nombres hexadécimaux de plus de quatre éléments dans le CPC 464, mais souvent vous n'utiliserez que des nombres de deux éléments.

Le CPC 464 peut convertir le décimal en hexadécimal avec la fonction HEX$.

Par exemple, essayez d'entrer :

```
PRINT HEX$(16)
```

dans l'ordinateur. Vous lirez 10 sur l'écran (une fois 16 et pas d'unités).

Le CPC 464 accepte les nombres hexadécimaux, mais ils doivent être précédés de & ou de &H pour indiquer à l'ordinateur qu'il s'agit d'hexadécimal. Cette caractéristique permet la conversion hexadécimal/décimal.

Par exemple :

```
PRINT &FF
```

vous donnera 255.

OPÉRATIONS LOGIQUES

Les mots **Basic, AND *(et)* et OR *(ou)*** peuvent pratiquement être utilisés comme dans le langage courant, nous l'avons vu dans le chapitre précédent. Le *ou* exclusif (XOR) a aussi été

brièvement traité dans un chapitre antérieur en tant qu'opérateur logique. En fait, ces trois opérateurs logiques peuvent être utilisés en mode binaire, et c'est un sujet important que nous allons considérer en détail.

En général, dans ce mode, deux octets sont comparés bit à bit pour donner un octet comme réponse. Si vous donnez à l'ordinateur des nombres décimaux, l'opération vous donnera des résultats qui sembleront à première vue aberrants. Essayez de taper cette commande dans le CPC 464 :

```
PRINT 15 AND 245
```

vous obtiendrez 5 et non pas 260 comme on pouvait l'attendre. Si nous convertissons ces trois nombres en binaire, on aura le résultat suivant

15	00001111
245	11110101
5	00000101

L'ordinateur compare en réalité les deux nombres bit à bit, 1 est inscrit dans le résultat si les deux bits de la colonne valent 1. Il y a deux 1 dans la colonne des unités, c'est pourquoi 1 apparaît dans le bit des unités. Prenons la colonne la plus à gauche ; il y a un seul 1 dans cette colonne, c'est pourquoi le bit de gauche du résultat vaut 0.

Vous n'y verrez peut-être qu'un intérêt académique, mais il y a des applications pratiques de ce type d'opération logique. La fonction AND est particulièrement utile en tant que masque, lorsque seuls certains bits sont intéressants et que les autres doivent être éliminés. Pour cela, on crée un nombre de masque avec des 1 pour les bits intéressants et 0 pour les bits à masquer. Pour garder les deux bits les moins significatifs d'un nombre, on utilise 3 comme masque, puisque ces deux bits correspondent respectivement à 1 et 2, qui donnent un total de 3. L'exemple suivant montre la manière avec laquelle le masque opère :

85	01010101
3	00000011
1 (réponse)	00000001

Un bit positionné à 0 dans le masque permet de s'assurer que le résultat aura un bit correspondant qui vaudra aussi 0. Si un bit du masque vaut 1, le résultat sera 0 ou 1, cela dépendra du nombre à masquer, mais le bit du résultat recopiera le bit du nombre considéré.

La fonction OR opère de manière similaire. Un 1 apparaît dans le résultat, s'il y a un 1 dans la colonne correspondante (ou bien s'il y en a deux). Ainsi, si la fonction OR opère sur 85 et 3, on a le résultat suivant :

85	01010101
3	00000011
87 (résultat)	01010111

La fonction XOR opère également de la même manière, mais un 1 apparaît dans le résultat seulement si un bit vaut 1 dans la colonne correspondante ; 1 n'apparaît pas si les deux bits sont à 1. Voici le résultat de 85 et 3 par cette fonction :

85	01010101
3	00000011
86 (résultat)	01010110

Chapitre 9

INTERFACE

De nombreux problèmes d'interface propres aux micro-ordinateurs ont pu être évités par le CPC 464, qui possède un lecteur de cassette et un moniteur monochrome ou couleur (les deux, si l'on possède les éléments de connection nécessaires). Des accessoires propres à l'Amstrad, comme les manettes, sont utilisés avec la machine. Elles se branchent aisément dans la prise appropriée. Bien sûr, l'emploi du lecteur de cassettes n'évite pas tous les problèmes associés à ce type d'enregistrement, et il est nécessaire d'utiliser des cassettes de bonne qualité, et de garder les têtes de lecture en bon état, de manière à obtenir des résultats de bonne qualité. L'emploi du mode standard (1 000 bauds) au lieu du mode rapide (2 000 bauds) donne une qualité optimale.

IMPRIMANTE

Le CPC 464 possède un interface parallèle d'impression (type Centronics), qui permet de l'utiliser avec une grande gamme d'imprimantes, des moins chères aux plus perfectionnées. Si cela est possible, nous recommandons l'emploi d'adaptateurs déjà réalisés, même si vous êtes familiarisés avec l'électronique et la soudure, car cela vous épargnera du temps et des désillusions. S'il est difficile d'obtenir un tel adaptateur, ou bien si vous préférez le réaliser, il faudra utiliser un câble de 23 voies, d'une longueur d'un mètre à peu près (mais pas plus de deux mètres) pour relier un connecteur approprié à un Centronics 36 voies, chaque broche du connecteur étant reliée à la broche correspondante du Centronics. Le diagramme de l'appendice V du manuel du CPC 464 donne la numérotation des broches de l'accès imprimante, et le connecteur du Centronics à des broches numérotées. Remarquons que certaines broches ne

sont pas connectées (celles données par l'appendice V du manuel le sont). Notons qu'il n'y a pas de ligne vérifiant la transmission sur l'accès imprimante. Pour remédier à cela, une impulsion négative est envoyée sur la ligne *strobe* lorsqu'un caractère valable est transmis sur les huit lignes de données (D0 à D7), ceci pour indiquer à l'imprimante que la donnée est bonne. L'imprimante indique alors sur la ligne *acknowledge* ou *Busy handshake*, qu'elle a fini de traiter la donnée, et qu'elle est prête à recevoir un autre caractère. Seule, l'une des deux lignes *Acknowledge* ou *Busy handshake*, doit être connectée, et non pas les deux ensemble. L'absence de données sur la ligne *acknowledge* n'est pas importante puisque l'utilisateur en tient compte, et utilise la ligne *Busy handshake*.

Un petit problème peut survenir lorsque l'on essaie de relier une imprimante : obtenir un connecteur adapté à l'accès imprimante. Un connecteur 2 fois 17 par pas de 0,1 pouce de type femelle doit être utilisé ; mais un tel connecteur est peut-être difficile à obtenir. Il existe une solution simple : acheter un connecteur plus large, et lui donner la taille requise, à l'aide d'une scie à métaux.

Un dernier point : bien qu'il existe huit voies pour les données, D7 est connecté au sol. C'est possible pour les codes ASCII qui utilisent des nombres inférieurs à 127, et qui donc n'emploient pas D7. Cependant, certaines imprimantes peuvent traiter des nombres compris entre 128 et 255, mais ils ne peuvent être transmis à l'accès imprimante.

IMPRESSION

Les programmes peuvent être listés sur une imprimante, à l'aide de la commande LIST, mais avec le nombre 8 qui caractérise l'imprimante, au lieu d'utiliser LIST seul, (qui inscrit sur l'écran). Ainsi la commande

```
LIST#8
```

donnera le programme entier sur l'imprimante. Si seules les

lignes 10 à 100 doivent être listées, il suffit alors de spécifier ces lignes dans l'instruction (par exemple LIST 10–100,# 8).

On peut aussi imprimer (PRINT) des données sur l'imprimante en utilisant la commande PRINT avec le nombre 8 (par exemple PRINT#8, « Ceci sera inscrit sur l'imprimante »).

Une commande utile qui ne doit pas être négligée est WIDTH. Elle donne le nombre maximum de caractères par ligne (ceci tant que cette valeur est inférieure au nombre maximum permis sur l'imprimante). Par exemple, la commande WIDTH 25 limitera la longueur de la ligne à 25 caractères.

ACCÈS DISQUETTE

L'accès disquette a été initialement conçu pour être utilisé avec un lecteur de disquettes Amstrad, mais il peut servir, en fait, comme interface dans beaucoup d'autres emplois. En particulier, il constitue un interface idéal pour des utilisateurs multiples de l'appareil. Pour quelqu'un d'expérimenté, il est assez facile d'utiliser cet accès, mais il faut souligner qu'il est imprudent de s'en servir si l'on n'a pas l'expérience suffisante. Cela pourrait causer des dégâts onéreux sur l'ordinateur. Les informations ci-dessous, concernant cet accès, s'adressent aux lecteurs qui possèdent cette expérience.

On peut obtenir sur cet accès, les données, l'adresse, les bus de contrôle, ceci avec une tension de 5V (qui doit être capable de supporter au moins 100 milliampères). La connection à cet accès se fait par l'intermédiaire d'un connecteur 2 fois 25 par pas de 0,1 pouce ; un tel connecteur est facilement réalisable à partir de connecteurs plus grands. La méthode standard d'interface avec un appareil utilisant un microprocesseur Z 80 consiste à décoder seulement les huit lignes d'adresses basses (A0 à A7) du bus d'adresses ; on obtient ainsi 256 adresses d'entrée ou de sortie.

Elles ont des adresses distinctes, et sont placées dans des zones entrée/sortie. MEMRQ est au niveau bas lorsque l'on accède à la mémoire, IORQ est au niveau bas lorsqu'une

instruction d'entrée/sortie apparaît. En plus du bus d'adresse, IORQ doit être décodé pour s'assurer que des circuits supplémentaires n'agissent pas lorsqu'on accède à un circuit mémoire de même adresse.

De même que beaucoup de micro-ordinateurs ayant le Z80, le CPC 464 n'applique pas complétement la méthode standard d'interface du Z80. Le système adopté est d'utiliser les huit lignes d'adresses hautes (A8 à A15) pour sélectionner les circuits d'entrée/sortie désirés ; les lignes d'adresses basses étant réservées pour le cas où un circuit d'entrée/sortie demanderait plus d'une adresse. L'intérêt de ce système est de permettre de réduire considérablement l'emploi du nombre d'adresses codées, ce qui facilite et simplifie le *hardware*. Le seul problème de ce système apparaît lorsqu'une adresse fausse est émise ; alors plusieurs circuits d'entrée/sortie sont activés en même temps. Il est cependant peu probable que cela cause des dégâts ou même un arrêt du système, mais il peut en résulter des interruptions ou des distorsions lors de l'inscription. Il faut cependant vérifier, lorsque l'on adresse directement une instruction entrée/sortie, si elle est destinée à un circuit interne ou externe. Lorsque l'on utilise le code machine, seules les instructions d'entrée/sortie, dans lesquelles le registre B donne les huit bits de poids élevé du bus d'adresse, peuvent être employées, et les macro-instructions qui se servent du registre B comme compteur doivent être évitées.

CIRCUITS EXTERNES

La taille de la zone-mémoire d'entrée/sortie qui est libre pour les périphériques, s'étend de &F800 à &FAFF. Cependant, il n'est pas nécessaire de décoder tout le bus d'adresse. Le système de base de ce décodage, suggéré par Amstrad, est d'activer des circuits extérieurs lorsque la ligne d'adresse A10 passe en position basse. S'il faut plus d'une adresse d'entrée/sortie, on peut en utiliser jusqu'à 256 en décodant l'ensemble ou quelques-unes des huit lignes d'adresses basses avec A10.

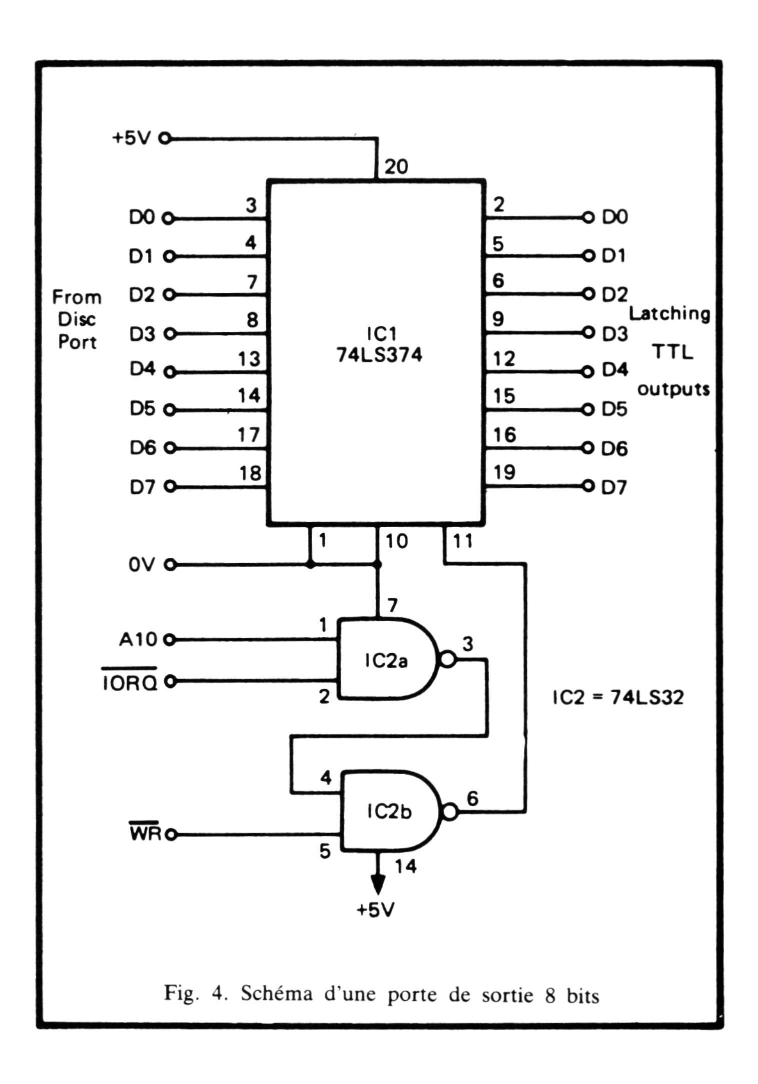


Fig. 4. Schéma d'une porte de sortie 8 bits

La figure 4 montre comment une porte de 8 bits (verrouillage) peut être reliée à l'accès-disquette. IC1 est un composant de type *flip/flop* à 3 états de sortie. Dans notre exemple, il est utilisé comme verrouillage 8 bits lorsque le décodeur d'adresse envoie un signal négatif sur l'entrée-horloge qui correspond à la broche 11 de IC1, ceci bien sûr lorsqu'une donnée est disponible sur l'accès disquette. Le décodeur d'adresse est très simple, puisqu'il comprend seulement deux portes appartenant au composant 741532 disposées de sorte qu'une impulsion négative de verrouillage survienne lorsque A10, IORQ et WR (ligne d'écriture) sont en position basse.

Les données peuvent être écrites sur l'accès disquette en Basic en utilisant l'instruction OUT, accès à l'adresse &F800. En fait, l'emploi du décodage partiel d'adresse signifie que l'accès donne les adresses de sortie en numérique mais &F800 est une adresse simple et pratique à utiliser. Voici un exemple simple d'écriture de données sur l'accès.

```
OUT &F800,255
```

Toutes les lignes de sortie seront alors en position haute.

La figure 5 montre un circuit supplémentaire utilisé comme porte d'entrée avec possibilité de verrouillage. IC2 *a*, représenté sur la figure 4 associé à une porte inutilisée de IC2 (IC2c), est utilisé pour décoder les lignes A10, IROQ et RD *(read)*. De la sorte, une impulsion négative est envoyée sur IC4 lorsqu'une adresse &F800 d'entrée/sortie est lue. Le décodage des lignes WR et RD permet d'aiguiller la même adresse sur les portes d'entrée/sortie, sans que les deux circuits interfèrent entre eux. IC4 est un verrou 8 bits transparent. Lorsque la ligne *strobe* de gauche n'est pas utilisée, il est identique à une porte 8 bits qui transmet intégralement l'information. Si la ligne *strobe* est au niveau bas, une impulsion positive peut verrouiller la donnée dans la porte. Ceci peut être intéressant si l'accès est commandé par une source de signal, comme un convertisseur analogi-

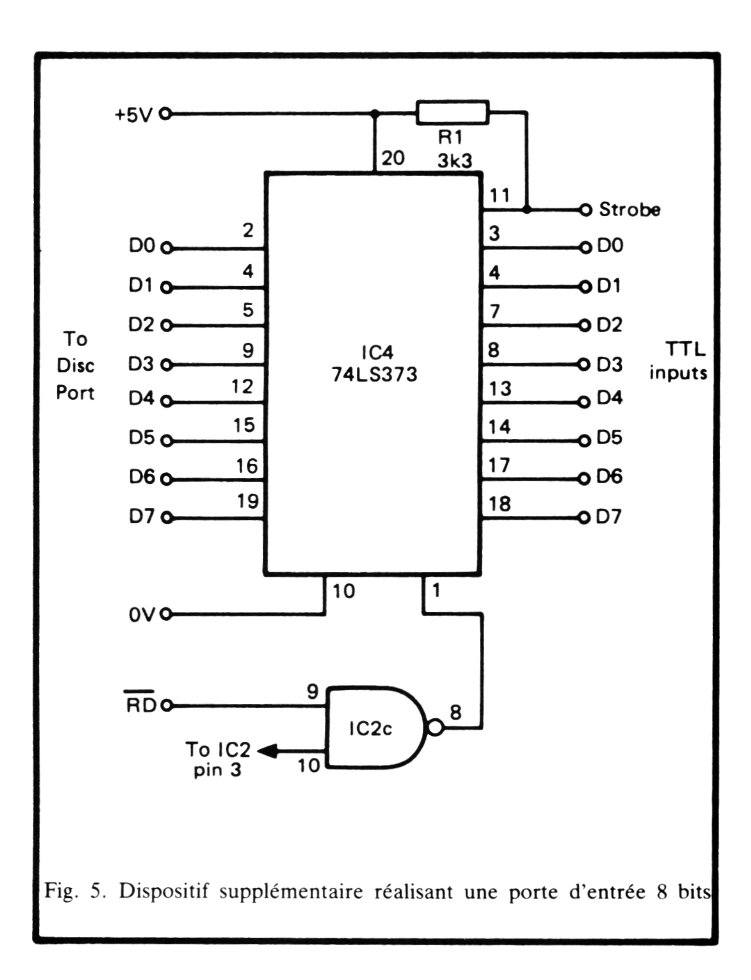


Fig. 5. Dispositif supplémentaire réalisant une porte d'entrée 8 bits

que numérique qui travaille en mode de conversion continu. Le circuit peut être modifié de sorte que le convertisseur verrouille le résultat de chaque conversion dans la porte. On lit alors le contenu de la porte, qui donne la dernière indication du convertisseur ; ceci évite d'autre part l'emploi des lignes *handshake*.

La porte peut être lue en Basic à partir de la fonction INP.

Par exemple : PRINT INP (&F800)

liera la porte et indiquera la valeur lue sur l'écran.

Les circuits illustrent la méthode d'interface de base de l'accès-disquette. Cette méthode peut être utilisée avec d'autres circuits, en particulier les périphériques Z80A tels que le Z80A PIO. Comme le CPC 464 utilise la version rapide (4MHZ) du Z80 (le Z80A), tous les périphériques Z80 employés avec l'ordinateur doivent être des versions A rapides.

L'accès disquette comporte une voie-son qui est une entrée audio. Il est nécessaire de disposer de 50 millivolts RMS pour pouvoir l'utiliser. Bien sûr il y a une sortie stéréo, disponible sur la palette arrière de l'ordinateur. Les connections sont réalisées avec cette sortie par l'intermédiaire d'une prise stéréo de 3,5 mm (celle utilisée pour les casques stéréo).

Cette sortie peut être utilisée pour commander un amplificateur audio, mais le niveau disponible n'est pas suffisant pour alimenter un casque. Il existe aussi une entrée de luminance *(light pen registers)* sur l'accès disquette. Les registres du 6845 CRT, qui comprennent les registres de luminaire, peuvent être lus en écrivant le numéro du registre dans l'adresse &8C00, puis en lisant l'adresse d'entrée/sortie &8F00. Les registres de luminance sont les registres 16 et 17. Ils ne donnent pas directement les positions horizontales et verticales sur l'écran, mais ils sont mélangés pour former un nombre de 16 bits qui est alors traité pour correspondre aux coordonnées de l'écran.

PEEK ET POKE

Nous avons étudié l'instruction OUT et la fonction INP, que l'on emploie lors d'opérations entrée/sortie. On doit aussi mentionner les instructions POKE (enregistre l'octet) et PEEK (donne le contenu de l'octet en décimal), qui sont leurs équivalents lorsque l'on utilise des circuits mémoire. Dans beaucoup d'ordinateurs, ces deux instructions sont très importantes, mais la structure et le logiciel interne du CPC 464 sont tels que vous n'aurez pas à les employer très souvent. Le CPC 464 dispose de 64 K de RAM (mémoire vive) qui correspondent aux adresses du Z80A allant de 0 à 65535. Vous pouvez employer PEEK et POKE pour chacune de ces adresses, mais il faut noter que l'emploi de POKE pour certaines adresses peut tromper l'ordinateur (mais vous ne pouvez causer de dégâts en utilisant PEEK ou POKE). Certaines adresses, en fait la moitié, correspondent à de la RAM (où les programmes et les données sont enregistrées) et de la ROM (qui contient le logiciel interne — Locomotive Basic — qui est le système de travail de l'ordinateur). Seule, la RAM est accessible par les insructions PEEK et POKE.

PORT DE SORTIE

Lorsque l'accès imprimante n'est pas employé en tant que tel, on peut l'utiliser comme sortie à verrouillage 8 bits. La donnée disponible sur ce port est écrite à l'adresse &EF00. Les bits 0 à 6 étant représentés par D0 à D6 respectivement, sur le connecteur de l'accès imprimante. Comme on le sait déjà, D7 est relié à la masse, et le bit 7 est disponible sur la sortie *strobe*. Cependant ce bit est inversé alors que les 7 autres ne le sont pas. Si l'on veut, ce bit peut être réinversé, si c'est nécessaire.

Il est possible de lire l'entrée *Busy*, qui est le bit 6 de l'adresse d'entrée &F500. Il est aussi possible de lire un seul bit d'un port en masquant les autres bits ; pour cela on utilise la fonction AND, comme on l'a vu précédemment.

Il existe une instruction Basic qui permet de sélectionner

un, ou plusieurs bits d'un port d'entrée : c'est l'instruction WAIT. Dans sa forme la plus simple, elle indique le ou les bits de l'adresse donnée, et boucle jusqu'au moment où le bit (ou bien l'un des bits) passe à l'état haut. Par exemple :

```
WAIT &F800,3
```

bouclera, jusqu'à ce que le bit 0 ou le bit 1 du port d'entrée (qui se trouve à l'adresse &F800) passe à l'état haut. En fait, cette instruction réalise un AND entre le nombre contenu dans l'adresse &F800 et le masque donné dans l'instruction. L'ordinateur répète cette procédure jusqu'au moment où l'on obtient une réponse différente de zéro.

Cette instruction peut prendre une autre forme si l'on introduit un troisième paramètre. On réalise alors un AND avec le masque, puis un XOR* avec le troisième paramètre. Le but de cette instruction est de pouvoir faire boucler un programme, jusqu'à ce qu'un bit, ou plusieurs, passent à l'état bas, au lieu de l'état haut, et ceci parce que XOR (*ou* exclusif) inverse la valeur du ou des bits.

Par exemple :

```
WAIT &F800,3,3
```

sera réalisée, tant que l'un des bits 0 à 1 du port d'entrée adressé par &F800 passera à l'état bas au lieu de l'état haut.

* XOR désigne le *ou* exclusif.

Chapitre 10

INTERRUPTIONS

Une caractéristique de Locomotive Basic est son aptitude à générer des interruptions, ou, au moins, une forme d'interruptions. Au sens strict, les interruptions sont créées par des procédés *hardware* internes à l'ordinateur, qui activent l'arrêt de l'accès des données au microprocesseur. La réalisation d'une horloge est une application classique des interruptions. Dans ce cas, un circuit *hardware* génère à intervalles réguliers une interruption, typiquement 100 fois par seconde (300 fois par seconde, dans le cas du CPC 464). Un logiciel associé au circuit incrémente le nombre qui est rangé dans une série de mémoires de un, à chaque fois qu'une interruption est générée par le circuit.

Lorsqu'on lit ce nombre dans les mémoires, on obtient une horloge, qui peut éventuellement être utilisée comme horloge temps réel si l'on procède lors de la lecture à quelques opérations mathématiques. La plupart des micro-ordinateurs utilisent beaucoup les interruptions pour contrôler le clavier, par exemple, ou bien synchroniser les voies sonores.

La particularité des interruptions est qu'elles ne sont normalement pas visibles par l'utilisateur de l'ordinateur. Si l'on travaille en mode commande, et que l'on exécute un programme, les interruptions continuent d'être générées. Si elles ne sont pas vues, c'est parce que l'intervalle de temps est très court. Lorsqu'une interruption est générée, le microprocesseur finit de traiter l'instruction présente, puis considère la procédure d'interruption, et l'applique jusqu'à ce qu'elle soit retirée. Les interruptions créées à partir du Basic sur le CPC 464 sont analogues aux interruptions réelles, mais différentes tout de même, puisqu'il s'agit de procédés logiciels et non pas structurels. Cependant, elles peuvent encore continuer à être générées alors qu'un programme est exécuté. C'est leur seule possibilité d'utilisa-

tion, car elles ne peuvent être employées en mode commande.

AFTER

Deux instructions servent à générer des interruptions : AFTER et EVERY. Considérons, tout d'abord AFTER : il crée une interruption après qu'un certain temps soit passé. En pratique, un sous-programme est associé à chaque instruction AFTER. Lorsque l'intervalle de temps s'est écoulé, le programme s'arrête, et se rend au début du sous-programme indiqué. Après l'exécution de celui-ci, le programme retourne là où il s'était arrêté. Si c'est nécessaire, on peut utiliser jusqu'à quatre instructions AFTER en même temps, chaque instruction occupant un canal différent (0, 1, 2, 3). Celui que l'on utilise par défaut est le numéro 0. Le petit programme ci-dessous va vous aider à comprendre l'instruction AFTER.

```
 10 AFTER 50,0 GOSUB 100
 20 AFTER 100,1 GOSUB 200
 30 AFTER 150,2 GOSUB 300
 40 AFTER 200,3 GOSUB 400
 50 PRINT "Le programme boucle"
 60 GOTO 50
100 PRINT "Interruption 0"
110 RETURN
200 PRINT "Interruption 1"
210 RETURN
300 PRINT "Interruption 2"
310 RETURN
400 PRINT "Interruption 3"
410 RETURN
```

Les quatre premières lignes de ce programme génèrent quatre interruptions AFTER qui déterminent le temps qui doit s'écouler avant chaque interruption, le canal déterminé, et le sous-programme que l'on doit appeler (dans cet ordre).

L'intervalle de temps qui précède l'interruption est calculé en cinquantième de seconde. Les lignes 50 et 60 servent à faire boucler le programme indéfiniment, "Le programme boucle" étant inscrit sur l'écran. Les quatre sous-programmes sont écrits des lignes 100 à 410, ils impriment simplement « interruption » suivi du numéro du canal qui génère l'interruption. Si vous exécutez ce programme, vous trouverez sur l'écran, parmi les lignes « Le programme boucle » des messages « interruption 0 », « interruption 1 » et ainsi de suite qui apparaissent après les temps spécifiés (1, 2, 3 et 4 secondes).

EVERY

L'instruction EVERY diffère de AFTER, en ce sens que le sous-programme est appelé régulièrement. Le premier nombre donné par l'instruction détermine de temps qui doit s'écouler avant que le sous-programme soit exécuté, ainsi que la période entre deux exécutions. Ce nombre est aussi exprimé en cinquantième de seconde.

Le listing ci-dessous illustre l'instruction EVERY

```
 10 x=0
 20 EVERY 50 GOSUB 100
 30 WINDOW 1,34,40,1,1
 40 PRINT "Le programme boucle"
 50 GOTO 40
100 x=x+1
110 PRINT 1,x
120 RETURN
```

La ligne 10 initialise la variable x à la valeur 0, alors que la ligne 20 appelle le sous-programme, qui débute en 100, toutes les secondes. Notons que lorsqu'on utilise AFTER et EVERY, le sous-programme doit se terminer par RETURN, comme tout sous-programme appelé par GOSUB. Ce sous-programme imprime simplement un nombre dans une fenêtre (WINDOW) (voir ligne 30) en haut à gauche de

l'écran. Ce nombre vaut initialement 1, et est incrémenté de 1, lorsque le sous-programme est appelé. En d'autres termes, nous avons créé un compteur de secondes. Les lignes 40 et 50 constituent une boucle, qui imprime répétitivement "Le programme boucle", sur l'écran.

Si vous exécutez ce programme, le compteur de secondes apparaît en haut à gauche de l'écran, mais chaque nombre n'apparaît qu'un instant, puisqu'aucune commande qui empêcherait le déroulement sur l'écran (et ainsi la suppression des nombres) n'a été placée.

DI (suppression d'interruptions) et EI (permission d'interruptions) sont deux instructions associées aux interruptions temporelles. Comme leurs noms l'indiquent, elles sont utilisées pour supprimer les interruptions, afin d'éviter un arrêt dans la procédure, puis pour les réautoriser, une fois la procédure terminée. REMAIN est une fonction liée aux interruptions, elle imprime le compte à rebours du canal déterminé (par exemple REMAIN (2) indique le nombre de cinquantièmes de secondes qu'il reste, avant qu'une interruption EVERY ou AFTER n'apparaisse sur le canal 2). Il est important de noter que la fonction REMAIN laisse le compteur à 0 et s'oppose à l'appel des sous-programmes. Comme les instructions EVERY et AFTER utilisent les quatre n^{ièmes} compteurs, il est impossible d'envoyer simultanément sur le même canal, une instruction AFTER et une instruction EVERY. Le nombre maximum possible est 32 767, ce qui représente un délai de 11 minutes, largement suffisant pour la plupart des applications pratiques.

Notes

Notes

Notes

Notes

Notes

Notes

Notes

LA BIBLIOTHÈQUE EDIMICRO

• *Collection « Ordinateurs familiaux »*

VG 5000

Amsler, Bardon	**Guide du VG 5000 Philips**
Amsler, Villemaud	**Jeux sur VG 5000 Philips**

MO5

Bieber, Perbost, Renucci	**Tout sur le MO5**
Perbost, Renucci	**Jeux sur MO5**

AMSTRAD

Penfold	**L'Amstrad avec plaisir**
Waugh	**Musique sur Amstrad**

ATMOS/ORIC 1

Kosniowsky	**Nouveaux Jeux sur ATMOS**
Chane-Hune, Darbois	**Guide de l'Oric**
Bayvejiel	**Jeux Graphiques sur ATMOS**
Viguier	**Premiers Pas en Programmation**

SPECTRUM

Bridge, Carnell	**Aventures sur Spectrum**
Hurley	**Jeux Graphiques sur Spectrum**

COMMODORE 64

Fleurier, Meiller	**Jeux d'adresse et de hasard**
Fleurier, Meiller	**Jeux d'action et de réflexion**

ELECTRON/BBC

Bennani, Chaieb	**Graphisme et sons sur Electron et BBC**

PHILIPS C7420 VIDEOPAC +

Bardon, de Merly	**Jeux sur Philips C7420 Vidéopac +**

TO7

— ouvrages

Bieber, Perbost, Renucci	**Guide du TO7**
Perbost, Renucci	**Jeux sur TO7**

— logiciels sur cassette

Perbost, Renucci	**4 jeux avec manette sur TO7**
Perbost, Renucci	**6 jeux d'action et réflexion sur TO7**

GENERAL

Djama	**Je construis mon premier robot**
Ly	**Applications familiales en Basic MSX**
Perbost, Berthet	**Introduction à MSX**
de Merly	**Ordinateur familial : Que choisir ?**

• *Collection « Ordinateurs professionnels »*

APPLE II, IIe, IIc

de Merly	**GUIDE DE L'APPLE** **Tome 1 : l'Apple standard** **Tome 2 : les extensions** **Tome 3 : les applications**

MACINTOSH

Gaucherand .... **MACINTOSH : outils, progiciels, applications**

SINCLAIR QL

Tenin, Van Thong .... **Guide pratique du Sinclair QL**
K et S Brain .... **Intelligence artificielle sur QL**

• ***Collection «Progiciels»*** *animée par MM. Bonnet et Dinh*

APPLE II, IIe, IIc

Bonnet, Dinh .... **Multiplan sur Apple : exercices de gestion**

IBM PC

Bonnet, Dinh .... **Lotus 1-2-3 à votre portée**
Bonnet, Dinh .... **Multiplan sur IBM PC : exercices de gestion**
Hœnig .... **Framework à votre portée, avec exemples**
Commandeur .... **Framework sur IBM PC : exercices de gestion**

MACINTOSH

Bouilloux .... **Multiplan et Chart sur Macintosh**
Bouilloux .... **Excel sur Macintosh : exercices de gestion**
Bouilloux .... **CX MacBase 500 : exercices de gestion**
Hornn .... **Jazz sur Macintosh : exercices de gestion**

SINCLAIR QL

Tenin .... **Archive: bases de données sur Sinclair QL**
Devisscher .... **Abacus, Easel et Quill : calcul, graphisme et texte sur QL**

GENERAL

Bonnet, Dinh .... **Mémento Multiplan**

• ***Collection «Langages»***

Gaucherand, Lamoitier .... **Fichiers en Basic par l'exemple**
Longevialle .... **Pascal sur Macintosh**

• ***Collection «Ouvrages de base»***

Lamoitier .... **Le Traducteur Micro**
Darnis, Van Thong .... **Graphisme et CAO**
Avon, Rolet .... **Courbes de maths en Basic**

Nombreux autres titres à paraître. Catalogue sur simple demande.

EDIMICRO 121-127, avenue d'Italie, 75013 Paris

Achevé d'imprimer en décembre 1985
sur les presses de l'imprimerie Laballery et Cie — 58500 Clamecy
Dépôt légal : décembre 1985 N° d'imprimeur : 511076

L'AMSTRAD AVEC PLAISIR

Comment faire de bons programmes

R.A. et J.W. PENFOLD

LE COMPAGNON INDISPENSABLE DU PROGRAMMEUR SUR AMSTRAD CPC 464, pour apprendre à écrire des programmes de qualité, et à progresser avec aisance vers des réalisations plus subtiles.

- Les auteurs suivent une démarche pas à pas, en partant du plus élémentaire jusqu'aux techniques avancées. De nombreux exemples de programmes illustrent et clarifient les notions abordées.
- Au sommaire : Variables et tableaux — Variables chaînes — INPUT, PRINT et DATA — Décisions — Le générateur de sons — Le graphisme et l'animation — Le binaire et l'hexadécimal — Les interfaces et les interruptions.
  Ce livre au format de poche, complet et pratique, vous aidera vraiment à tirer le maximum de votre Amstrad.

ISBN : 2-904457-31-3 59 F.



121-127, Avenue d'Italie 75013 Paris

Edimicro

L'AMSTRAD AVEC PLAISIR

R.A.-J.W. PENFOLD



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/